Practical Threshold Elliptic Curve Cryptography From Native Assumptions

Yashvanth Kondi

Acknowledgements

Firstly, I would like to thank abhi shelat, my PhD advisor. abhi gave me his time and energy (and grant funding) to help me forge the skills that I sought, and produce work of which I am proud. I am indebted to abhi for being unfailingly supportive of me throughout my PhD, and an incredible mentor overall.

Next, I would like to thank my thesis committee: Claudio Orlandi, Daniel Wichs, and Jon Ullman. I first met Claudio at a conference in India when I was but an undergrad, and I was bemused (and terrified) that he took my babbling seriously. Our subsequent interactions through my PhD have been of great benefit to me. Daniel created a wonderful atmosphere in which to learn cryptography at Northeastern through his excellent classes and reading groups, as well as informal conversations each time we met. My interactions with Jon have been informative and entertaining as well.

I was introduced to the wizardry that is modern cryptography in a fantastic course taught by Ashish Choudhury at IIIT-Bangalore in 2015. Thereafter, he made several opportunities available to me, one of the most significant being to participate in cryptography research at Arpita Patra's lab at the Indian Institute of Science. I am very grateful to both Ashish Choudhury and Arpita Patra for giving me my start in cryptography—both in terms of skills, as well as practical matters. I'd also like to acknowledge my labmates at the CrIS lab at IISc during my time spent there: Ajith Suresh, Divya Ravi, Pratik Sarkar, Swati Singla, Megha Byali, and numerous others with whom my visits overlapped. I was also lucky to meet Chaya Ganesh when she was visiting IISc, as she has since been a source of great career advice, and a valuable collaborator.

Even further back in 2013, I learnt one of my most important lessons for research in general during an internship at a non-profit in Bangalore called Fields of View, thanks to Sruthi Krishnan and Niveditha Menon. They taught me to identify and question my own biases, and that it's possible to absorb ideas that fundamentally challenge my understanding of the world. I slowly learnt not to dismiss (or be threatened by) ideas that did not align with my own intuition, which I found particularly difficult to do within the confines of the Indian education system, especially given my caste and gender privilege. In addition to training my

mind to internalize new ways of thinking—technical and beyond—I believe that these skills have enabled me to better contextualize my research.

At Northeastern, I was fortunate to have a wonderful cohort with whom to share my PhD journey. Eysa Lee, Jack Doerner, and Schuyler Rosefield featured prominently throughout the years—we shared most classes, many meals, multiple papers, numerous discussions, and an advisor (just the one). Willy Quach, Ariel Hamlin, Matthew Jagielski, and Giorgos Zirdelis completed the crypto PhD student group for the bulk of my time at Northeastern, and were excellent colleagues and friends through and through. Ran Cohen, a postdoc (and later research scientist) at Northeastern was generous with his time when I needed it, and played an important role in my career development through his mentorship. The past postdocs and visitors of the crypto group, newer crypto students, and the rest of the theory group at Northeastern were also vital in creating a pleasant research atmosphere: Megan Chen, Siyao Guo, Mor Weiss, Chethan Kamath, LaKyah Tyner, Ethan Mook, Biswaroop Maiti, Tanay Mehta, Lydia Zakynthinou, Akshar Varma, Vikrant Singhal, Chin Ho Lee, and several others. There are of course many more from the Northeastern community who made my time there enjoyable, not least the ISEC crowd that inevitably congregated in the coffee area each afternoon. Especially worth mentioning is Ching Daranuwat, whose restaurant (Bangkok Pinto) kept all of us nourished with reliably delicious Thai cuisine throughout the five years. I am also grateful to the Khoury administrative staff, especially Sarah Gale and Laura Adrien.

Much of the work in this thesis is the outcome of two research visits that I was lucky to make during my PhD—one in person, and one virtual during the pandemic. The former was a visit to Aarhus University during the summer of 2019, where I was generously hosted by Claudio Orlandi. Besides being productive research wise, this visit exposed me to the wholesome and constructive dynamic of the Aarhus crypto group. I was lucky to have met many group members who made my visit enjoyable: Bernardo Magri, Daniel Escudero, Akira Takahashi, Eduardo Soria-Vazquez, Pierre Meyer, Sophia Yakoubov, Ivan Damgård, Peter Scholl, and several others. Omer Shlomovits was a great collaborator during this time.

The latter visit was an internship in the summer of 2020 at the Novi division of Facebook/Meta. Lera Nikolaenko was my supervisor, and an exceptional one at that; besides proactively acquiring the right resources for me, she helped me develop an understanding of how engineering constraints influence cryptographic design. In addition to Lera, I learnt a lot from François Garillot, Kostas Chalkias, and Payman Mohassel during the internship. My co-interns Jasleen Malvai, Panos Chatzigiannis, and Yan Ji were very good company, despite us only having met online.

Arriving in a new country and finding one's feet is always difficult, and I was extremely lucky to have had family in the United States to make the transition easier. Sirish Kondi and

Avanti Badami helped me with a lot of the practicalities, from my first meal on arriving in the US, to getting a working phone and literally paying my first few bills. Venkatesh Tanjore and Mani Lakshmi Appalla provided for me a slice of home away from home, and visiting D.C. was a treat each time thanks to their hospitality. Sahana, Srini, and Akash Purohit gave me a warm welcome to Boston, and were lovely hosts each time I visited their home. Shruthi Nagaraj and Madhu Rao were wonderful hosts as well, and their proximity to the Boston area was a source of comfort.

I am deeply grateful to my family—my parents K.R. and Radha Mohan, and my sister Shruti Kondi—for being supportive of my choices in all aspects of my life. My parents worked hard to provide all that I needed to succeed in my career, while also ensuring that I enjoyed my childhood and college days. Shruti has been a source of encouragement through the years, and a generally high quality sibling. Lloyd, Angela, David, Michelle, Vikram, Pippin, and Sums Nazareth were a supportive presence during the past few years, and I greatly appreciate them giving me a wholehearted welcome into their family.

I'd like to acknowledge my cousins Sirish and Sushanth Kondi for sparking in me an early interest in science, and my grandfather Srinivasa Murthy for inducing my appreciation of mathematics. The rest of the Sukhamayanivas gang deserves a special mention for their support and good humour—notably Bhagya and K.R. Ananda Rao, Deepti Nagaraj, Apeksha and Akanksha Sushanth Kondi. My friends from college were a great source of camaraderie through the years, especially Ananth Murthy, Akshay Jindal, and Chandan Yeshwanth.

Finally, and possibly most significantly, I would like to acknowledge my partner Anisha Nazareth's role in my development, both personally, and as a researcher. As of writing, it has been exactly ten years since we first met, and it is difficult to overstate how instrumental she has been in my growth through these years. Indeed, her prolific and superb baking has been conducive to my growth in multiple dimensions. Anisha is a talented researcher and an insightful conversationalist, and has on many occasions helped me uncover blind spots and biases in my own views. Simultaneously challenging someone hampered by imprecise ideology (or blinded by privilege), while being respectful of their dignity (and accommodating of their sometimes frustratingly slow self-discovery) is no mean feat, and yet Anisha has accomplished this with grace and unwavering faith in me on every occasion. More materially, Anisha has been immensely supportive on innumerable occasions—taking extensive care of me after an unexpected surgery, hosting many a grand meal for my friends, and helping me finish crosswords to name a few. As a person, her companionship has empowered me to navigate the hurdles I've faced with confidence. As a researcher, she has taught me to appreciate nuance and grapple with complexity—and as any cryptographer will tell you, complexity and hard problems are not a bad thing.

Perhaps unsurprisingly, this thesis is dedicated to Anisha.

Contents

List of Tables

List of Figures

List of Acronyms

- AES – Advanced Encryption Standard
- CRS – Common Reference String
- DL/DLog – Discrete Logarithm
- ECDSA – Elliptic Curve Digital Signature Algorithm
- EdDSA – Edwards Curve Digital Signature Algorithm
- GC – Garbled Circuit
- MPC – Secure Multiparty Computation
- NIZK – Non-interactive Zero-knowledge
- NP – Non-deterministic Polynomial Time
- OT – Oblivious Transfer
- PoK – Proof of Knowledge
- PPT – Probabilistic Polynomial Time
- PRF – Pseudorandom Function
- PRNG – Pseudorandom Number Generator
- RO – Random Oracle
- SHA – Secure Hashing Algorithm
- SNARK – Succinct Non-interactive Argument of Knowledge
- UC – Universal Composability

- WH – Witness Hiding
- WI – Witness Indistinguishability
- ZK – Zero-knowledge

Abstract

Threshold cryptography is the practice of distributing the secret keys of a cryptosystem across multiple devices, so that a consortium of these devices must collaborate in order to use the key. This makes threshold cryptography a useful tool for cryptographic key management, as it can mitigate single points of failure in a cryptosystem by decentralization.

In this thesis, we study how to securely decentralize signing with the most commonly used elliptic curve based signature schemes—ECDSA and EdDSA—and develop novel protocols that are efficient to operate in common parameter ranges, while being sensitive to real-world constraints. The constraints we consider include interaction, reliable access to entropy, and state continuity. To this end, we make progress on the fronts of non-interactively aggregating EdDSA signatures, proactive threshold ECDSA state refresh with offline devices, and derandomizing threshold EdDSA signing with native resilience to state resets.

Our results on EdDSA signature aggregation include a highly efficient scheme that achieves rate 2 compression, albeit with a loose security proof, and a rate $2-o(1)$ compressing scheme (with tight security) that is slower but tolerable for many applications. The latter construction is enabled by improving upon the state of the art for straight-line extraction in the random oracle model. We further improve on the state of the art by expanding the applicability of Fischlin's transformation, and proving (and matching, for some parameters) a lower bound on the Prover's random oracle query complexity.

In the context of long-term security for threshold ECDSA, we study a notion of proactive state refresh with 'offline' devices, where the signing consortium can refresh the state of the system and leave update packages for offline devices to catch up at their leisure. We give an efficient construction to achieve this notion for $(2, n)$ threshold ECDSA, and show a fundamental barrier to its immediate extension.

Finally, we explore the problem of derandomizing threshold EdDSA signing while achieving natural resilience to state resets. We show via a new construction that such provably secure resilience can be achieved at lower latencies than heuristic solutions that rely on trusted hardware. We develop new techniques for committed oblivious transfer, and garbling the exponentiation function in order to achieve this result.

All of our constructions use tools native to the signature schemes that they decentralize, and their practicality is justified either empirically or by concrete analysis.

Chapter 1

Introduction

In the nineteenth century, Dutch linguist and cryptographer Auguste Kerckhoffs formulated one of the first principles of modern cryptography:

> *The security of a system must be uncompromised even if an adversary were to learn the design of the system.*

This concept is widely realized today by making public any cryptographic algorithms that constitute a system, but keeping a randomly sampled "key" hidden from view. Maintaining secrecy of the keys is critical for most deployed cryptosystems today; if an attacker were to obtain the key, the system might be rendered completely insecure.

Given the total dependence of a cryptosystem upon its key, any device that stores such a key is potentially a single point of failure for the security of the entire system. Single points of failure are highly undesirable for any secure architecture, as evidenced by the increasing adoption of multi-factor authentication as industry standard practice [Wha21].

A Solution Paradigm: Decentralization. The generation and storage of cryptographic key material can instead be distributed across multiple devices in such a way that an adversary trying to steal the key is forced to break into more than one device. A suite of accompanying protocols can allow the devices to collaboratively perform cryptographic operations (such as signing) without ever having to reconstruct the key at a single physical location. Under the assumption that security failures for the devices are reasonably uncorrelated—such as by using independent software stacks and hardware—decentralization offers a promising solution to the single point of failure problem for cryptographic keys. This approach of *threshold cryptography* was first conceived by Desmedt [Des88].

Practical Modern Multiparty Signing. In the context of digital signatures, threshold cryptography implies a distributed system in which the secret key is generated and accessed for signing only via multiparty protocols. The task of designing such a system can be viewed as an instance of Secure Multiparty Computation (MPC). Fundamental decades old results on MPC have already established the *theoretical* feasibility of securely distributing any task [Yao86, GMW87, BGW88] including multiparty signing. With a sophisticated theory supplying a foundation, MPC and multiparty signing are primed to find their place as standard tools in system designers' toolkits. However as with any nascent technology that is in the process of being forged from an applied science, several challenges arise upon the introduction of real-world constraints. In this thesis, we develop multiparty signing protocols for *commonly deployed elliptic curve schemes* that function efficiently under such constraints. The particular constraints that are of interest to us are those of *interaction*, *long-term security*, *resilience to poor entropy*, and *statelessness*.

1.1 Problem Scope

With the above constraints in mind, we introduce the specific domains that we study in this thesis.

The security of multiparty signing protocols is typically conditional on no more than a threshold number of devices in the system being under adversarial control. While this may be a reasonable expectation in the short term, in the long term it is prudent to plan for the eventual compromise of every device—under the assumption that fewer than a threshold number are compromised at any given time. We study **proactive state refresh** in this thesis as one such long-term defense mechanism. In particular, we study proactive state refresh under a specific interaction constraint, which allows the state of the system to be rerandomized with the same consortium of parties that typically convenes to produce a signature.

Another avenue for an adversary to induce a security failure in the lifetime of a system is to exploit vulnerabilities relating to randomness generation in otherwise uncompromised devices. In particular, entropy is usually a scarce resource, and many deployments instead draw their randomness from stateful random number generators. However, avoiding state reuse (and therefore randomness reuse) in the face of power supply interruptions, Virtual Machines instantiated with stale snapshots, unsafe restoration from backups, etc. is a notoriously difficult problem to solve in practice. In this thesis, we study **distributed signing that is stateless and deterministic by design** in order to avoid this issue altogether.

In many settings, the scale and nature of parties involved imply that it is highly impracti-

cal to have them all run an interactive protocol in order to generate a joint signature. While the trivial approach to capture the effect of their joint signing is to have them sign individually and simply concatenate their signatures, we investigate **non-interactive signature aggregation** as a method to achieve some compression of the trivial approach.

Which Signature Algorithms to Decentralize? The most commonly used public key cryptosystem across the internet today is RSA [RSA78], although elliptic curve based schemes are rapidly increasing in popularity due to the improved operational efficiency that comes with their smaller key sizes [Hen22]. The Elliptic Curve Digital Signature Algorithm (ECDSA) is the most widely deployed elliptic curve based signature scheme, followed by the more recent Edwards-curve Digital Signature Algorithm (EdDSA). This surge in the use of these elliptic curve cryptosystems elicits research that facilitates their efficient and safe deployment. Indeed, as of the writing of this thesis, the ongoing NIST Multiparty Threshold Cryptography project [NIS] seeks to develop guidelines for their decentralization, and serves as an indicator of where industry and government interest lies. We therefore focus on ECDSA and EdDSA in this thesis.

Better Building Blocks for Threshold Signatures. Protocols to distribute signing for ECDSA/EdDSA frequently make use of tailored cryptographic proofs (with or without zero-knowledge) as building blocks, particularly to prove knowledge of discrete logarithms in the corresponding elliptic curve groups. As contributions that may be of independent interest in this thesis, we improve the state of the art in instantiating such primitives, under the umbrella of **straight-line extraction in the random oracle model**.

1.1.1 The Tradeoffs We Make

The term "practical" is in itself highly sensitive to context, and the notion of a practical scheme is constructed by accounting for various environmental factors. Most secure computation tasks have a plethora of options for modern cryptographic tools with which to address them, and each comes with its own performance profile and security guarantees. Choosing from amongst these cryptographic tools is informed by priorities in efficiency metrics and other contextual concerns.

For the settings relevant in this thesis we determine computation cost to be a priority, and when constructing a multiparty protocol for signing with a given signature scheme, we use cryptographic tools native to the signature scheme itself. Using cryptographic tools native to the signature scheme—in particular the same hash function and elliptic curve—minimizes

the trusted base of complexity assumptions as well as systems dependencies (eg. by the reuse of verified libraries) while maintaining compatibility with widely accepted standards.

Why Optimize Computation Cost? We briefly justify here why minimizing the computation cost (at the expense of bandwidth for instance) is a meaningful tradeoff, and defer a more nuanced discussion to the relevant technical sections. In the case of multiparty signing for ECDSA, consider two representative applications: one of mobile devices where signing is human-initiated, where the important metric is the latency of producing a signature, and the other of a consortium of servers signing many messages, where the important metric is throughput. Assuming realistic connectivity, our benchmarks [DKLs18, DKLs19] indicate that for multiparty ECDSA signing, both metrics are determined by computation cost; the latency due to computation on mobile devices is higher than that of data transmission over 4G LTE, and the servers exhaust computational resources before saturating a gigabit connection. These claims are further evidenced by independent benchmarks due to Gavenda [Gav21] for the mobile setting, and Dalskov et al. [DOK+20] for the server setting.

Within the scope of the stateless deterministic signing problem, our goal is to show that addressing the problem at the protocol design level while providing better security guarantees than a heuristic trusted hardware approach, need not come at a performance penalty. In particular, our point of comparison is the use of special purpose hardware with a "trusted monotic counter" that is incremented each time a signing operation is executed. In our work [GKMN21] we estimate that the latency induced by maintaining such a counter can be outperformed only by the most computationally lightweight of cryptographic mechanisms to enable stateless deterministic signing.

In the signature aggregation setting, if we consider the naive concatenation of signatures to be our baseline, then *any* time required for aggregation must be tolerable latency for the higher level application. In a system that involves hundreds or thousands of signers—such as a large scale multisignature, or cryptocurrency spenders per block in a blockchain, where the system latency is a few minutes or tens of seconds—it may be reasonable for aggregation to require a few seconds, but beyond that the aggregation operation itself may induce unacceptable latency for the system. While using generic succinct proofs off the shelf can achieve $\omega(1)$-rate compression, for EdDSA and instantiations of Schnorr that use standardized cryptographic functions, the induced aggregation time is on the order of minutes at least. In contrast, our $\theta(1)$-rate compression scheme [CGKN21, Ks22] can aggregate thousands of signatures in under a second.

In essence, across all of the settings that we consider to be relevant for our work, the present state of affairs suggests that the conservation of computational resources is a meaningful

objective.

1.2 Summary of Results

With the context of distributed signing for ECDSA and EdDSA established, we now proceed to give an informal overview of the results contained in this thesis. We begin with our work on non-interactive signature aggregation in Section 1.2.1, and then the closely related problem of straight-line extraction in the random oracle model in Section 1.2.2. Subsequently we discuss our contributions to proactive long-term security in Section 1.2.3, and finally stateless deterministic signing in Section 1.2.4.

1.2.1 Non-interactive EdDSA Signature Aggregation

In many scenarios, signers may not be able to interactively collaborate (at any point) in producing signatures. Such settings include authorities that issue certificate chains in TLS, large scale consensus systems where validator nodes may be too numerous to speak to each other, and cryptocurrency spenders whose transactions are to be included in a common block. An *aggregate* signature [BGLS03] allows any public third party to compress the effect of a number of independent signatures into a single object, without the signers having to interact with anyone (including the aggregator). While pairing-based signatures are known to permit such compression without interaction [BLS01, BGLS03, BDN18] they are not as widely implemented as ECDSA or EdDSA. Unfortunately signature schemes based on the discrete logarithm problem alone are not known to natively permit non-interactive aggregation. One could of course use generic compressing proof machinery [Gro16, BBHR18, BBB+18], but the high proving cost (i.e. large circuit sizes) induced by statements that involve standard hash functions like SHA2—as used by ECDSA/EdDSA—would likely be prohibitive for most applications.

We therefore initiate the study of *native* non-interactive aggregation techniques for discrete logarithm based signature schemes. In particular, we constrain ourselves to aggregation that is blackbox in both the hash function as well as the group (in which discrete logarithms are assumed to be hard to compute) used by the signature scheme. In this section, we will use the term "native aggregation scheme" to mean a "non-interactive proof of knowledge of $n \in \mathbb{Z}$ signatures, where the prover and verifier make blackbox use of the hash function and group of the signature scheme".

On the positive side, we construct a public coin three-move protocol to prove knowledge of a collection of Schnorr signatures, where the proof transcript is only half the size of the con-

catenation of all the signatures [CGKN21]. Applying the Fiat-Shamir transformation [FS87] to this protocol yields a highly efficient non-interactive proof—essentially an aggregated signature that achieves 50% compression—which we empirically validate with the EdDSA scheme as incurring essentially no computational penalty (in proving/verification time).

Theorem 1.2.1. *(Informal) There is a native aggregation scheme for Schnorr signatures in the random oracle model, where the bit length of the aggregate signature is half of the naive concatenation of signatures.*

On the negative side, we also show that this is essentially the best possible compression that one can hope to achieve given the constraints, in particular when the prover's algorithm must be blackbox in the hash function.

Theorem 1.2.2. *(Informal) No native aggregation scheme for full-entropy Schnorr signatures (where the hash function used for the signature is treated as a random oracle) can achieve a compression rate better than* $2 + o(1)$.

To our knowledge, there are no techniques to prove knowledge of Schnorr signatures that are non-blackbox in the Schnorr hash function, which do not involve expressing the hash function as an arithmetic circuit and invoking general SNARKs. Our theorem therefore serves as evidence that achieving sublinear aggregation of *standardized* Schnorr signatures is inherently computationally expensive, i.e. tied to the SNARK proving complexity of hash functions that are believed to be correlation intractable as relevant for Schnorr [CCH+19].

Provably Secure Parameterization. An unfortunate consequence of using the Fiat-Shamir transformation is reliance upon the Forking Lemma [PS96] for the security proof, which incurs a quadratic loss in the reduction of security of the aggregated signature to that of the underlying signature scheme itself. Besides being problematic for concurrent composition, this means that in order to provably retain the same security level, the underlying Schnorr signatures must be doubled in size, which eliminates the 50% compression due to aggregation. This quadratic loss comes from a proof technique that *rewinds* the adversary in order to extract a witness. One can instead obtain a *tight* proof (and thus tight parameters) with only a small loss in the compression rate by using *straight-line extraction* techniques, for instance by applying Fischlin's transformation [Fis05] instead of Fiat-Shamir.

Theorem 1.2.3. *(Informal) There is a native aggregation scheme for Schnorr signatures that achieves a compression rate of* $2 - o_\kappa(1)$*, whose security tightly reduces to unforgeability of the underlying Schnorr scheme.*

The o_κ notation treats terms dependent on the security parameter (i.e. κ) as constants, as is common in the SNARK literature when describing how performance scales with witness size [BCR+19]. By tight security, we refer to the fact that the extraction error for the proof of knowledge is bounded by $2^{-\kappa}$. However upon benchmarking the resulting construction, we found the computational cost of aggregation by applying Fischlin's transformation to be prohibitively high for many applications.

We therefore perform a detailed study of straight-line extraction techniques, making improvements to the security, applicability, and efficiency of the state of the art. For the signature aggregation setting in particular, we improve performance by two orders of magnitude so it may be tolerable for many applications [Ks22]. We summarize our contributions to this area below.

1.2.2 Improved Straight-Line Extraction

In the context of zero-knowledge proofs of knowledge in the random oracle model, a *straight-line extractor* is able to produce a witness simply by reading the queries that the prover made to the random oracle in the production of the proof, which eliminates the need to rewind the prover and enables tight reductions. We revisit Fischlin's transformation [Fis05], which upon application to a Sigma protocol produces a non-interactive proof of knowledge with straight-line extraction in the random oracle model. A constraint of this transformation is that it only applies to Sigma protocols that support 'quasi-unique responses', which informally means that it is computationally hard to find more than one accepting response to a challenge. At a high level, Fischlin's Prover must solve a 'proof of work' problem in order to produce a proof, in particular by finding accepting Sigma protocol transcripts of the form (a, e, z) such that $H(a, e, z) = 0$ for an appropriate hash function H. A folklore belief regarding Fischlin's work [Fis05, pg. 13] is that the transformation also extends to languages where a statement may have multiple witnesses, for eg. logical combinations of Sigma protocols [CDS94]. We show via an attack that this belief is false, and tighten the conditions under which the transformation applies by means of a new proof [Ks22].

Theorem 1.2.4. *(Informal) Fischlin's transformation does not preserve Witness Indistinguishability when applied to the Sigma protocol to prove knowledge of one out of two discrete logarithms.*

Intuitively, the problem stems from the deterministic nature of Fischlin's prover, which we show how to address by a careful randomization mechanism.

Theorem 1.2.5. *(Informal) Any* strong *special sound sound Sigma protocol can be compiled to a non-interactive zero-knowledge proof of knowledge with the same bandwidth and*

computation efficiency as Fischlin's technique.

Strong special soundness—a notion that we introduce—requires that any pair of valid Sigma protocol transcripts that share a first message can be used to extract a witness. This is stronger than special soundness (which has the additional constraint on the transcripts that they must not share the same challenge) and in conjunction with our randomization technique, replaces Fischlin's use of quasi-unique responses.

Computational Cost. With signature aggregation as our motivating application, we make progress in the study of the computation cost in producing straight-line extractable proofs in the random oracle model. The cost of the Schnorr/EdDSA aggregation protocol [CGKN21] can be characterized as having to evaluate a degree n polynomial at T points, where T corresponds to the number of random oracle queries required to solve Fischlin's proof-of-work problem. We improve both components: we show that T can be reduced by changing the underlying proof-of-work problem to that of finding *collisions*, and we improve the cost of evaluating the polynomial by applying an $O(n^{1.5})$ algorithm that is uniquely suited to the parameters relevant to signature aggregation. We note that improving on the naive $O(n^2)$ polynomial evaluation algorithm is non-trivial, as the commonly used Fast Fourier Transform is incompatible with signing curves deployed in practice, and general asymptotically superior polynomial evaluation algorithms [vzGG13, BCKL21] are not efficient for the relatively low degrees relevant to the applications for signature aggregation. Combined with a tighter analysis of parameters for the proof of work problem, we empirically demonstrate an improvement of two orders of magnitude over the naive application of Fischlin's transform to aggregate EdDSA signatures for practically relevant parameters (up to 1000 signatures at a time), and obtain a performance envelope that is tolerable for many applications.

The idea of improving T (which we call the *prover query complexity*) by using a collision-based proof of work carries over to the more general zero-knowledge setting as well. We show that Fischlin's NIZKPoK prover can be sped up by 11-15% by applying this idea, and for a special class of Sigma protocols (that structurally resemble the signature aggregation setting) by up to a factor of ≈ 2. While this is a modest improvement, it is significant as it meets (for the first time, for non-trivial parameters) a new lower bound on prover query complexity that we present.

Lemma 1.2.6. *(Informal) If a NIZKPoK scheme for a hard relation with a straight-line extractor (in the non-programmable ROM) induces a verifier to make V queries to the RO for a κ-bit security level, then the prover must on average make at least $P_{\mathsf{OPT}}[V, \kappa] = (V! \cdot 2^{\kappa})^{\frac{1}{V}}$ queries in generating a proof.*

Our lower bound is a tightening of an asymptotic lower bound by Fischlin [Fis05, Proposition 2].

1.2.3 Long-Term Security

Most practical multiparty computation protocols are designed to tolerate *static* adversaries (i.e. corruptions fixed prior to execution), typically for efficiency purposes. However in the long term, external security failures may render this model inadequate. One such scenario is where the adversary is able to compromise all devices in the system through its lifetime, but never more than a threshold at any given point in time. Ostrovsky and Yung [OY91] were the first to consider this model of *mobile* adversaries, and formulated a defence mechanism termed *proactive security* in subsequent work by Canetti and Herzberg [CH94]. The method of proactive secret sharing due to Herzberg *et al.* [HJKY95]—which laid the template for most subsequent work [MZW+19]—involves all devices in the system periodically coming together in order to interactively rerandomize their shared state. However certain inadequacies of such mechanisms are brought out in applications relating to cryptocurrency custody.

In the case of cryptocurrencies, even accidental loss of the secret key corresponds to a total loss in the funds associated with the key. Therefore it is imperative to design mechanisms that facilitate key recovery, without inducing too much extra risk due to redundant storage. In this regard, a $(2, 3)$ threshold architecture is known to be a reasonable configuration [Eya21]. For instance, two 'online' devices are invoked to sign messages (via a threshold signing protocol), and an 'offline' cold storage server keeps a third share and can assist with key recovery in case one of the online devices fails. This complicates proactive share refresh, as an explicit goal is to keep the cold storage server isolated from the signing devices. We therefore formulate a model of 'offline refresh' [KMOS21] in which any subset of parties required to access/operate on the shared secret is also sufficient to rerandomize the state of the system, and any offline devices that don't participate in this rerandomization can 'catch up' after the fact. Formalizing the task itself is subtle—we settle on a notion that we call *unanimous erasure*, which can be seen as the proactive analogue of the standard multiparty computation notion of 'security with unanimous abort' [FGH+02].

Achieving unanimous erasure even in the $(2, 3)$ case is non-trivial, intuitively because of the inherent unfairness of two-party computation [Cle86] combined with the fact that private channels are not verifiable—this may create a situation where an honest online party is unable to obtain an update message to send the offline server, while its corrupt counterparty may (or may not) have already sent such a message via its private channel to the offline server to induce a share refresh.

As a first step, we make use of a public ledger to construct an offline refresh mechanism for $(2, n)$ threshold Schnorr/EdDSA signatures, via a novel *interleaved threshold signing* technique. Our use of the ledger is immediately compatible with the way that existing cryptocurrency wallets already use the blockchain.

The ECDSA scheme is generally more challenging to decentralize due to its non-linear signing equation. Threshold ECDSA can be constructed using two-party multiplication as a building block [DKLs18, DKLs19, LN18, GG18]. We proceed with an instantiation based on Oblivious Transfer (OT), as it uses tools native to the ECDSA signature, while being computationally light [DKLs18, DKLs19]. We then devise a proactivization technique for the state that it induces via Beaver's OT preprocessing method [Bea95]. We subsequently construct a signing protocol for $(2, n)$ threshold ECDSA with support for offline refresh that extends our Schnorr techniques, and show empirically that our refresh mechanism adds reasonable computational overhead (14%–24%) to existing threshold ECDSA protocols.

Finally, we show that our definition of offline refresh is too strong to satisfy for (t, n) threshold schemes (when $t > 2$) in a model that allows a random oracle and an ideal ledger, by means of a novel proof.

Theorem 1.2.7. *(Informal) For any integers $n > t > 2$, it is impossible for a (t, n) threshold signature scheme to achieve resilience to $t - 1$ malicious corruptions while also permitting an offline refresh protocol (with unanimous erasure) with only $2t - 2$ online devices.*

1.2.4 Stateless Deterministic Signing

Another issue to consider for the long-term security of a system is state reuse upon a reset. In particular, during the lifetime of a system, downtime can be incurred due to scheduled maintenance or even adversarial attacks. Restarting the system (eg. by reloading a Virtual Machine with a snapshot) may resume its operation from a stale state, which can be problematic for cryptographic applications as we discuss below. General solutions to this problem at the systems level are constructed under the umbrella of 'state continuity' [PLD+11, SP16, MAK+17], however they require additional physical assumptions such as helper servers or slow, expensive trusted hardware.

Most threshold signing protocols do not critically rely on updating state as written, however they do rely on a supply of fresh uniform randomness. As good entropy is scarce in practice, this randomness is typically obtained from a pseudorandom number generator. Reusing state in this context is equivalent to reusing randomness, which in turn is known to entirely void any security guarantees of ECDSA/Schnorr—in fact even a few bits of bias in their nonce generation is known to permit complete leakage of their secret keys [HS01, Hen22].

In the single signer setting where the signing key is available in its entirety this issue is easily solved by deriving nonces as the result of applying a PRF on the message to be signed, an approach that is explicitly specified by EdDSA [BDL+12]. This makes signing *deterministic*, and *stateless* in that state need not be updated (nor fresh randomness sampled) after the one-time key generation phase.

We study how to extend stateless deterministic signing to multiparty Schnorr, as the naive application of the single-party technique is known to fail in the multiparty case [MPSW19]. Our resulting dishonest majority threshold Schnorr scheme [GKMN21] makes use of novel gadgets for garbled circuits and Committed Oblivious Transfer.

Theorem 1.2.8. *(Informal) There is an n-party threshold Schnorr protocol where signing does not require fresh randomness nor the ability to update state. The protocol makes blackbox use of the group and hash function used to instantiate Schnorr (but non-blackbox use of a PRF), and the public key operations that a party is induced to perform when signing a message is $O(n)$ exponentiations.*

We estimate that for several useful real-world parameters, our stateless deterministic threshold signing protocol will induce a lower latency than using trusted hardware for general state continuity.

1.3 Thesis Organization

In Chapter 2 we establish the notation, model, and tools with which our results are achieved. Chapter 3 presents the background for our results, along with an overview of our techniques. The subsequent chapters contain the technical details—Chapter 4 pertains to signature aggregation and straight-line extraction, Chapter 5 to derandomizing threshold signing statelessly, and Chapter 6 to achieving proactive security with a natural communication pattern. Finally, we give our concluding remarks in Chapter 7.

Chapter 2

Preliminaries

2.1 Notation

Throughout this paper, we use κ as the security parameter, and $(\mathbb{G}, G, q)$ to represent the elliptic curve over which signatures are calculated, where $\mathbb{G}$ is the group of curve points, G the curve generator, and q the order of the curve. Curve points are represented in $|q| = 2\kappa$ bits, where κ is the security parameter. Curve points are denoted with capitalized variables and scalars with lower case. We use $=$ for equality, $:=$ for assignment, $\leftarrow$ for sampling from a distribution, $\approx_c$ for computational indistinguishability, and $\equiv_s$ for statistical indistinguishability.

2.2 ECDSA

ECDSA is parameterized by a group $\mathbb{G}$ of order q generated by a point G on an elliptic curve over the finite field $\mathbb{Z}_p$ of integers modulo a prime p. Assuming a curve has been fixed, the ECDSA algorithms are as follows [KL15]:

Algorithm 2.2.1. $\mathsf{Gen}(1^\kappa)$

1. Uniformly choose a secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q$.
2. Calculate the public key as $\mathsf{pk} := \mathsf{sk} \cdot G$.
3. Output $(\mathsf{pk}, \mathsf{sk})$.

Algorithm 2.2.2. $\mathsf{Sign}(\mathsf{sk} \in \mathbb{Z}_q, m \in \{0,1\}^*)$

1. Uniformly choose an instance key $k \leftarrow \mathbb{Z}_q$.
2. Calculate $(R_x, R_y) = R := k \cdot G$.
3. Calculate
$$\sigma := \frac{H(m) + \mathsf{sk} \cdot R_x}{k}$$
4. Output $\sigma := (\sigma \mod q, R_x \mod q)$.

Algorithm 2.2.3. $\mathsf{Vrfy}(\mathsf{pk} \in \mathbb{G}, m \in \{0,1\}^*, \sigma \in (\mathbb{Z}_q, \mathbb{Z}_q))$

1. Parse σ as (σ, R_x).
2. Calculate
$$(r'_x, r'_y) = R' := \frac{H(m) \cdot G + R_x \cdot \mathsf{pk}}{\sigma}$$
3. Output 1 if and only if $(r'_x \mod q) = (R_x \mod q)$.

2.3 Schnorr/EdDSA

Like ECDSA, (elliptic curve based) Schnorr is parameterized by a group $\mathbb{G}$ of order q generated by a point G on an elliptic curve over the finite field $\mathbb{Z}_p$ of integers modulo a prime p. Assuming a curve has been fixed, the Schnorr generation algorithm is the same as ECDSA, and signing and verification are as follows:

Algorithm 2.3.1. $\mathsf{Sign}(\mathsf{sk} \in \mathbb{Z}_q, m \in \{0,1\}^*)$

1. Uniformly choose an instance key $k \leftarrow \mathbb{Z}_q$.
2. Calculate $(R_x, R_y) = R := k \cdot G$.
3. Calculate
$$\sigma := H(\mathsf{pk}, R, m) \cdot \mathsf{sk} + k$$
4. Output $\sigma := (\sigma \mod q, R_x \mod q)$.

Algorithm 2.3.2. $\mathsf{Vrfy}(\mathsf{pk} \in \mathbb{G}, m \in \{0,1\}^*, \sigma \in (\mathbb{Z}_q, \mathbb{Z}_q))$

1. Parse σ as (σ, R_x).
2. Calculate
$$(r'_x, r'_y) = R' := \sigma \cdot G - H(\mathsf{pk}, R, m) \cdot \mathsf{pk}$$
3. Output 1 if and only if $(r'_x \mod q) = (R_x \mod q)$.

The EdDSA scheme is essentially identical to the algorithm described above, with the exception that rather than uniformly sampling k, EdDSA samples a key sd for a PRF F during key generation, and computes $k = \mathsf{F}(\mathsf{sd}, m)$ when signing a message.

2.4 Standard Helper Functionalities

Several of the protocols in this thesis are written and proven in secure in the Universal Composability framework of Canetti [Can01]. In this section, we recall some standard functionalities necessary for the hybrid models in which we present our protocols.

We make use of UC Commitments, as described below.

Functionality 2.4.1. $\mathcal{F}_{\mathsf{Com}}$**. Commitments**

This functionality allows a sender S to commit to an indexed message, and reveal it at a later point in time. The sender's message remains secure iff an index is never reused for a different message. Additionally any index that is 'revealed' subsequently offers no security when reused. All messages are adversarially delayed.

Initialize: Wait for $(\texttt{init})$ from both parties, and store $(\texttt{init})$ in memory.

Commit: Upon receiving $(\texttt{commit}, \mathsf{ind}, m)$ from S, if $(\texttt{init})$ exists in memory,

- If R is not corrupt, store (ind, m) in memory and send $(\texttt{committed}, \mathsf{ind})$ to R
- Otherwise:
  1. If (ind, m') exists in memory such that $m \neq m'$ then send $(\texttt{reused-index}, m', m)$ to R
  2. If $(\texttt{index-used}, \mathsf{ind})$ exists in memory, then send $(\texttt{revealed-index}, \mathsf{ind}, m)$ to R
  3. If neither of the previous conditions hold, then store (ind, m) and send $(\texttt{committed}, \mathsf{ind})$ to R

Reveal: Upon receiving $(\texttt{reveal}, \mathsf{ind})$ from S, if (ind, m) exists in memory, then send $(\texttt{deommitted}, m)$ to R. Store $(\texttt{index-used}, \mathsf{ind})$ in memory.

We now describe a functionality to commit to a proof of knowledge of discrete logarithm.

Functionality 2.4.2. Committed ZKPoK for Discrete Log $\left(\mathcal{F}_{\mathsf{Com-ZK}}^{\mathsf{R}_{DL}}\right)$

This functionality is parameterized by the party count n and the elliptic curve $(\mathbb{G}, G, q)$. In each instance, one party P_i is the prover, and the others verify.

Commit Proof: On receiving $(sid, \texttt{com-proof}, x, X, \mathbf{I})$ from party P_i where $x \in \mathbb{Z}_q$ and $X \in \mathbb{G}$, if $(sid, \texttt{com-proof}, \cdot, \cdot, \cdot)$ does not exist in memory, then send $(sid, \texttt{committed}, i)$ to every party P_j for $j \in \mathbf{I}$ and store $(\texttt{com-proof}, x, X, \mathbf{I})$ in memory.

Decommit Proof: On receiving $(sid, \texttt{decom-proof})$ from party P_i, if there exists in memory a record $(sid, \texttt{com-proof}, x, X)$, then:

1. If $X = x \cdot G$, send $(sid, \texttt{accept}, X)$ to every party P_j for $j \in [n]$.
2. Otherwise send $(sid, \texttt{fail})$ to every P_j for $j \in [n]$.

Instantiating this functionality can be done with Schnorr's proof of knowledge of discrete logarithm Sigma protocol, plugged into any straight-line extractable sigma protocol to NIZK compiler in the random oracle model [Fis05].

Functionality 2.4.3. Prove all-but-one discrete logarithms $\left(\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{ZK}}^{\mathsf{R}_{DL}}\right)$

This functionality is parameterized by two parties P and V, and the elliptic curve $(\mathbb{G}, G, q)$. On receiving $(\texttt{prove}, I, (x_i)_{i\in I}, (X_i)_{i\in[n]})$ from P for an integer n and set of indices $I \subset [n]$ such that $|I| = n - 1$, $x_i \in \mathbb{Z}_q$, $X_i \in \mathbb{G}$, if $x_i \cdot G = X_i$ for each $i \in I$, then send $(\texttt{proven}, (X_i)_{i\in[n]})$ to V.

Instantiating this functionality can be done with Schnorr's proof of knowledge of discrete logarithm Sigma protocol in conjunction with the transformation of Cramer, Damgård and Schoenmakers to prove logical combinations of instances with Sigma protocols [CDS94], plugged into any general straight-line extractable sigma protocol to NIZK compiler in the random oracle model [Pas03]. This Sigma protocol even achieves Strong Special Soundness, as necessary for the NIZK compiler that we construct in this thesis.

$\mathcal{F}_{\mathsf{Coin}}$ This is a coin tossing functionality, which allows any pair of parties to publicly sample a uniform $\mathbb{Z}_q$ element.

Functionality 2.4.4. Coin Tossing ($\mathcal{F}_{\mathsf{Coin}}$)

This functionality is run with two parties P_0, P_1.

On receiving $(\texttt{sample-element}, \mathsf{id}^{\mathsf{coin}}, q)$ from both P_0, P_1, sample $x \leftarrow \mathbb{Z}_q$ uniformly and send $(\mathsf{id}^{\mathsf{coin}}, x)$ to both parties as adversarially delayed output.

Realizing this functionality is easy in the $\mathcal{F}_{\mathsf{Com}}$-hybrid model: P_0 samples $x_0 \leftarrow \mathbb{Z}_q$ and sends it to $\mathcal{F}_{\mathsf{Com}}$, following which P_1 samples $x_1 \leftarrow \mathbb{Z}_q$ and sends it to P_0. Finally P_0 instructs $\mathcal{F}_{\mathsf{Com}}$ to release x_0 and the output is defined as $x = x_0 + x_1$.

$\mathcal{F}_{\mathsf{MUL}}$ Secure two party multiplication functionality, in simplified form.

Functionality 2.4.5. Secure two party multiplication ($\mathcal{F}_{\mathsf{MUL}}$)

This functionality is run with two parties P_0, P_1.

On receiving $(\texttt{input}, \mathsf{id}^{\mathsf{coin}}, x_0)$ from P_0 and $(\texttt{input}, \mathsf{id}^{\mathsf{coin}}, x_1)$ from P_1 such that $x_0, x_1 \in \mathbb{Z}_q$, sample a uniform $(t_0, t_1) \leftarrow \mathbb{Z}_q^2$ conditioned on

$$t_0 + t_1 = x_0 \cdot x_1$$

and send t_0 to P_0 and t_1 to P_1 as adversarially delayed output.

For a more nuanced functionality that can be efficiently instantiated, along with such an instantiation based on Oblivious Transfer (which we describe how to proactivize in this thesis), we refer the reader to the work of Doerner et al. [DKLs19].

2.5 Sigma Protocols and Zero-knowledge

The standard definition of a Sigma protocol is given below.

Definition 2.5.1. *[Dam02] A Sigma protocol for relation R is a three move public coin protocol between a prover P_Σ and verifier V_Σ that has the following properties:*

- ***Completeness**: If P_Σ (with private input w) and V_Σ with public input x such that $(x, w) \in R$ execute the protocol honestly, then the protocol always terminates with V accepting.*
- ***Two-special soundness**: There exists an efficient extractor Ext which given as input the accepting conversations $T = (a, e, z)$ and $T' = (a, e', z')$ for statement x such that $e \neq e'$, outputs w such that $(x, w) \in R$.*

- ***Honest verifier zero-knowledge***: *There exists an efficient simulator $\mathcal{S}$ which upon input a statement x and challenge e outputs a, z such that (a, e, z) is an accepting conversation. Moreover when e is uniformly chosen, (a, e, z) is distributed identically to an execution of the honest protocol.*

A strong-special sound Sigma protocol—which is a notion that we introduce in this paper—additionally has the following property:

Definition 2.5.2. *A strongly two-special sound Sigma protocol for relation R is a three move protocol between a prover P and verifier V that is complete and honest verifier zero-knowledge as per Definition 2.5.1, and additionally has the following property:*

- ***Strong two-special soundness***: *There exists an extractor* Ext *which given as input the accepting conversations $T = (a, e, z)$ and $T' = (a, e', z')$ for statement x such that $T \neq T'$, outputs w such that $(x, w) \in R$.*

Next we present the definition of straightline extraction as given by Pass.

Definition 2.5.3 ([Pas03])**.** *We say that an interactive proof with negligible soundness (P, V) for the language $L \in NP$, with the witness relation R_L, is straight-line witness extractable in the RO model if for every PPT machine P^* there exists a PPT witness extractor machine E such that for all $x \in L$, all $y, r \in \{0,1\}^*$, if $P^*_{x,y,r}$ convinces the honest verifier with non-negligible probability, on common input x, then $E(view_V[(P^*x, y, r, V(x))], \ell) \in RL(x)$ with overwhelming probability, where $P^*_{x,y,r}$ denotes the machine P^* with common input fixed to x, auxiliary input fixed to y and random tape fixed to r, $view_V[(P^*_{x,y,r}, V(x))]$ is V's view including its random tape, when interacting with $P^*_{x,y,r}$, and ℓ is a list of all oracle queries and answers posed by $P^*_{x,y,r}$ and V.*

We recall Fischlin's transformation in Figure 2.1.

2.6 Garbling Schemes and Zero-knowledge

We first recall the syntax of garbled circuits, in the language of Bellare et al. [BHR12]. A garbling scheme $\mathcal{G}$ comprises: a garbling algorithm Gb that on input a circuit C produces a garbled circuit $\tilde{C}$ along with encoding information en and decoding information de. The encoding algorithm En maps an input x to a garbled input $\tilde{X}$ relative to en. The evaluation algorithm Ev then evaluates $\tilde{C}, \tilde{X}$ to produce a garbled output $\tilde{Y}$, which is then decoded by De using de to a clear output y. The verification algorithm Ve given $\tilde{C}, \mathsf{en}$ validates their well-formedness, and extracts the decoding information de if they are so.

Protocol $\pi_{\mathsf{NIZK}}^{\mathsf{Fis05}}$

The prover P and verifier V are both given the statement x while the prover also has a witness w for the statement $x \in L$. The security parameter κ defines the integers r, ℓ, t. These integers are related as $r \cdot \ell = 2^\kappa$, and $t = \lceil \log \kappa \rceil \cdot \ell$. Both parties have access to a Random Oracle $H : \{0,1\}^* \mapsto \{0,1\}^\ell$. The underlying sigma protocol is given by $\Sigma = ((\mathsf{P}_\Sigma^a, \mathsf{P}_\Sigma^z), \mathsf{V}_\Sigma)$.

$\mathsf{P}^H(x, w)$:

1. For each $i \in [r]$, compute $(a_i, \mathsf{state}_i) \leftarrow \mathsf{P}_\Sigma^a(x, w)$
2. Set $\boldsymbol{a} = (a_i)_{i \in [r]}$, and initialize $e_i = -1$ for each $i \in [r]$
3. For each $i \in [r]$, do the following:
   (a) If $e_i > t$, abort. Otherwise increment e_i and compute $z_i = \mathsf{P}_\Sigma^z(\mathsf{state}_i, e_i)$
   (b) If $H(\boldsymbol{a}, i, e_i, z_i) \neq 0^\ell$, repeat Step 3a
4. Output $\pi = (a_i, e_i, z_i)_{i \in [r]}$

$\mathsf{V}^H(x, \pi)$:

1. Parse $(a_i, e_i, z_i)_{i \in [r]} = \pi$, and set $\boldsymbol{a} = (a_i)_{i \in [r]}$
2. For each $i \in [r]$, verify that $H(\boldsymbol{a}, i, e_i, z_1) = 0^\ell$ and $\mathsf{V}_\Sigma(x, (a_i, e_i, z_i)) = 1$, aborting with output 0 if not
3. Accept by outputting 1

Figure 2.1: Fischlin's Transformation [Fis05]

For the purpose of the paper, we will assume that $\mathcal{G}$ is *projective* [BHR12], i.e. garbled input $\tilde{X} = (\mathsf{en}_{i,x_i})_{i \in [|x|]}$. We require the garbling scheme to be privacy-free [FNO15], i.e. satisfy two main security properties:

- **Authenticity**[1]: let $\tilde{C}, \mathsf{en}, \mathsf{de} \leftarrow \mathsf{Gb}(C, 1^\kappa)$ and $\tilde{X} \leftarrow \mathsf{En}(x, \mathsf{en})$, and $\hat{y} \neq C(x)$ for an adversarially chosen $C, x, \hat{y}$. It should be computationally infeasible for any PPT adversary $\mathcal{A}(\tilde{C}, \tilde{X})$ to output $\hat{Z}$ such that $\mathsf{De}(\mathsf{de}, \hat{Z}) = \hat{y}$.

[1] This is slightly weaker than the standard notion of authenticity [BHR12], which requires that *any* output other than $C(x)$ is hard to forge. It is sufficient for ZKGC if it is hard to forge an output only for any $\hat{y} \neq C(x)$ specified before $\tilde{C}, \tilde{X}$ are generated. Our gadget achieves this weaker notion, however it can easily be upgraded to the stronger notion if required by executing the gadget twice with independent randomness, and checking that they decode to the same output.

- **Verifiability**: given $\tilde{C}, \mathsf{en}$, the algorithm Ve produces decoding information de if $\tilde{C}$ is well-formed (i.e. a legitimate output of Gb). Alternatively if $\tilde{C}$ is malformed, Ve outputs $\perp$ with certainty.

Additionally we need 'Uniqueness', i.e. that if $C(x) = C(x')$, then $\mathsf{Ev}(\tilde{C}, \mathsf{En}(\mathsf{en}, x)) = \mathsf{Ev}(\tilde{C}, \mathsf{En}(\mathsf{en}, x'))$ for any valid $\tilde{C}, \mathsf{en}$. We give the formal definitions in Appendix B.3.

2.6.1 Committed Oblivious Transfer

Committed Oblivious Transfer (COT) offers the same interface as regular OT, but it also allows a 'reveal' phase where the both the sender's messages are revealed to the receiver, while the receiver's choice bit stays hidden. We encapsulate this notion (along with additional bookkeping to account for statelessness) in functionality $\mathcal{F}^*_{\mathsf{COT}}$. Additionally in order to facilitate a round compression optimization in the higher level protocol, $\mathcal{F}^*_{\mathsf{COT}}$ lets the sender lock its messages with a 'key', and reveals these messages upon the receiver presenting the key. We defer the formal details to Section 5.5.

2.6.2 Zero-knowledge from Garbled Circuits

We are now ready to recall a description of the original ZKGC protocol [JKO13]. The prover P holds a private witness x (of which the i^{th} bit is x_i), such that $C(x) = 1$ for some public circuit C.

1. The verifier V garbles the verification circuit, $\tilde{C}, \mathsf{en}, \mathsf{de} \leftarrow \mathsf{Gb}(C, 1^\kappa)$. Both parties engage in $|x|$ parallel executions of Committed Oblivious Transfer, with the following inputs in the i^{th} instance: V plays the sender, and inputs $\mathsf{en}_{i,0}, \mathsf{en}_{i,1}$ as its two messages. P plays the receiver, and inputs x_i as its choice bit in order to receive en_{i,x_i}.

2. P assembles $\tilde{X} = (\mathsf{en}_{i,x_i})_{i \in [|x|]}$ locally. V sends $\tilde{C}$ to P, who then computes $\tilde{Y} \leftarrow \mathsf{Ev}(\tilde{C}, \tilde{X})$, and sends $\mathsf{Commit}(\tilde{Y})$ to V.

3. V opens its randomness from all the COTs to reveal en in its entirety

4. P checks $\mathsf{Ve}(\tilde{C}, \mathsf{en}) = 1$, and if satisfied decommits $\mathsf{Commit}(\tilde{Y})$. V accepts iff $\mathsf{De}(\tilde{Y}, \mathsf{de}) = 1$

Intuitively the above protocol is sound due to authenticity of the garbling scheme: a malicious P^* who inputs x' such that $C(x') \neq 1$ to the OT will receive $\tilde{X}'$ such that $\mathsf{De}(\mathsf{Ev}(\tilde{C}, \tilde{X}'), \mathsf{de}) = 0$, and so to make V accept P^* will have to forge a valid $\tilde{Y}$ that is not the outcome of 'honest' garbled evaluation. Zero-knowledge comes from the verifiability and unique evaluation properties of the garbling scheme: an incorrect garbled circuit $\tilde{C}^*$ will be rejected in step 4

by P (who has not sent any useful information to V yet), and conditioned on $\tilde{C}$ being a valid garbled circuit, the uniqueness property hides which input was used to arrive at the output.

2.6.3 Extensions to ZKGC

The work of Chase et al. [CGM16] examines how to integrate proofs of algebraic statements into the garbled circuit based zero-knowledge framework, in order to prove composite statements. Roughly, their technique has P commit to a MAC of the witness $z = ax+b$ (computed via the garbled circuit/OT) along with $\tilde{Y}$ using a homomorphic commitment scheme. Once V reveals the randomness of the circuit, a, b become public and P leverages the homomorphism of the commitment in order to prove additional algebraic statements about the witness via Sigma protocols, such as the relation between x, z.

sequently Ganesh et al. [GKPS18] showed how to compress the original [JKO13] protocol to three rounds using a conditional disclosure of secrets technique, essentially by having V encrypt the OT randomness necessary for step 4 using the correct $\tilde{Y}$.

2.7 Proactive Security Notation and Assumptions

Throughout our exposition on proactive security, we fix the corruption threshold as $t = 1$ and hence formulate all of our definitions assuming one malicious adversarial corruption.

Network Model We assume a synchronous network, as already required by recent threshold signature schemes [GG18, LNR18, DKLs19]. For the blockchain model, we follow the synchronous functionality of Kiayias et al. [KZZ16]. In this functionality, the blockchain only progresses after all parties finish their current round, therefore parties are always synchronized during the protocol run.

Additionally, we make the necessary assumption of proactive channels that support delivery to offline parties, discussed further in Section 6.2.1.

Protocol Input/Output Notation for $(2, n)$ setting The $(2, n)$ proactively secure protocols in this thesis are described for any pair of parties indexed by $i, j \in [n]$. In particular, any two parties P_i, P_j out of a group of n parties $\mathbf{P}$ can run a protocol π with private inputs x_i, x_j to get their private outputs y_i, y_j respectively. For ease of notation since all of our protocols have the same instructions for each party, we choose to describe them as being run by P_b with P_{1-b} as the counterparty. The general format will be

$$y_b \leftarrow \pi(1 - b, x_b)$$

to denote that P_b gets output y_b by running protocol π with input x_b and counterparty P_{1-b}. For instance if π is run between P_2 and P_6, the protocol as described from the point of view of P_6 is interpreted with $b \equiv 6$ and $1 - b \equiv 2$.

Lagrange Coefficients $\lambda_i^j(x), \lambda_j^i(x)$ are the Lagrange coefficients for interpolating the value of a degree-1 polynomial f at location x using the evaluation of f at points i and j. In particular,

$$\lambda_i^j(x) \cdot f(i) + \lambda_j^i(x) \cdot f(j) = f(x) \qquad \forall x, i, j \in \mathbb{Z}_q$$

Each $\lambda_i^j(x)$ is easy to compute once i, j, x are specified.

2.8 Blockchain Model

We detail here the relevant aspects of the underlying blockchain system that is required for our proactively secure $(2, n)$ ECDSA protocol.

A Transaction Ledger Functionality A transaction ledger can be seen as a public bulletin board where users can post and read transactions from the ledger. As it was shown in [GKL15], a ledger functionality must intuitively guarantee the properties of *persistence* and *liveness*, that we informally discuss next.

- *Persistence*: Once a honest user in the system announces a particular transaction as *final*, all of the remaining users when queried will either report the transaction in the same position in the ledger or will not report any other conflicting transaction as stable.
- *Liveness*: If any honest user in the system attempts to include a certain transaction into their ledger, then after the passing of some time, all honest users when queried will report the transaction as being stable.

We encapsulate the ledger in a functionality $\mathcal{G}_{\mathsf{Ledger}}$ inspired by the functionality of [KZZ16].

On the Supported Type of Ledgers. For simplicity, we present our results on a synchronous public transaction ledger (e.g., Bitcoin [Nak09] or Ethereum [Woo]) where there is a known delay for the delivery of messages. We note however that synchrony of the ledger is not a necessary assumption for our protocol. In fact, any ledger satisfying the standard properties of *persistence* and *liveness* as defined in [GKL15] can be employed by our protocol. As it was shown in [GKL15], Bitcoin satisfies both properties for an honest majority of mining power under the assumption of network *synchrony*. However, if one is willing to trade off the

honest majority assumption for a partially synchronous network,[2] we point out that partially synchronous Byzantine Fault Tolerant (BFT) ledgers such as Algorand [CM19] can also be employed by our protocol due to how we define the corruption model, where the adversary "waits" for a full refresh before changing corruptions.

Without loss of generality we assume that every transaction that is included in the chain becomes final and will not be rolled-back. For a more detailed discussion we refer the reader to [BMTZ17]. We give a formal definition of the ledger functionality below.

Functionality 2.8.1. Global Ledger Functionality ($\mathcal{G}_{\mathsf{Ledger}}$)

The functionality $\mathcal{G}_{\mathsf{Ledger}}$ is globally available to all participants. The functionality is parameterized by a function $\mathsf{Blockify}$, a predicate $\mathsf{Validate}$, a constant T, and variables chain, slot, $\mathsf{clockTick}$ and buffer, and a set of parties $\mathcal{P}$. Initially set $\mathsf{chain} := \varepsilon$, $\mathsf{buffer} := \varepsilon$, $\mathsf{slot} := 0$ and $\mathsf{clockTick} := 0$.

- Upon receiving $(\texttt{Register}, sid)$ from a party P, set $\mathcal{P} := \{\mathcal{P}\} \cup P$ and if P was not registered before set $d_P := 0$. Send $(\texttt{Register}, sid, P)$ to $\mathcal{A}$.
- Upon receiving $(\texttt{ClockUpdate}, sid)$ from some party $P_i \in \mathcal{P}$ set $d_i := 1$ and forward $(\texttt{ClockUpdate}, sid, P_i)$ to $\mathcal{A}$. If $d_P = 1$ for all $P \in \mathcal{P}$ then set $\mathsf{clockTick} := \mathsf{clockTick} + 1$, reset $d_P := 0$ for all $P \in \mathcal{P}$ and execute *Chain extension*.
- Upon receiving $(\texttt{Submit}, sid, \mathsf{tx})$ from a party P, If $\mathsf{Validate}(\mathsf{chain}, (\mathsf{buffer}, \mathsf{tx})) = 1$ then set $\mathsf{buffer} := \mathsf{buffer}||\mathsf{tx}$.
- Upon receiving $(\texttt{Read}, sid)$ from a party $P \in \{\mathcal{A} \cup \mathcal{P}\}$, If P is honest then set $b := \mathsf{chain}$ else set $b := (\mathsf{chain}, \mathsf{buffer})$. Then return the message $(\texttt{Read}, sid, b)$ to party P.
- Upon receiving $(\texttt{Permute}, sid, \pi)$ from $\mathcal{A}$ apply permutation π to the elements of buffer.

Chain extension: If $|\mathsf{clockTick} - (\mathsf{T} \cdot \mathsf{slot})| > \mathsf{T}$ then set $\mathsf{chain} := \mathsf{chain}||\mathsf{Blockify}(\mathsf{slot}, \mathsf{buffer})$ and $\mathsf{buffer} := \varepsilon$, and subsequently send $(\texttt{ChainExtended}, sid)$ to $\mathcal{A}$.

The functionality $\mathcal{G}_{\mathsf{Ledger}}$ is parameterised by a set $\mathcal{P}$ of participants P; for a new participant to join the protocol it must send a message `Register` to the $\mathcal{G}_{\mathsf{Ledger}}$ functionality. We parameterise $\mathcal{G}_{\mathsf{Ledger}}$ by a constant T that denotes the gap in clock tick units between two

[2] It is a well known fact that it is impossible to achieve consensus on partially synchronous networks under honest majority [DLS88].

subsequent slots in the ledger. Without loss of generality, one could assume the existence of a function $\mathsf{Tick2Time}$ that maps clock ticks to physical time, in the same spirits of [KZZ16]. For concreteness, in such a case, the value of T would be 10 minutes in Bitcoin.

The functionality $\mathcal{G}_{\mathsf{Ledger}}$ is synchronous, and the $\mathsf{clockTick}$ variable is incremented only after all the parties send a message `ClockUpdate` to $\mathcal{G}_{\mathsf{Ledger}}$. A new block is created and appended to the chain only after T clock ticks have elapsed since the last block creation; in the meantime, parties can submit new transactions to the ledger with the message `Submit`, and read all the contents of the ledger with the message `Read`. The adversary $\mathcal{A}$ can permute the contents of the current transaction buffer, which translates to rearranging the order of the transactions that will be included in the next block.

We define the predicate $\mathsf{Validate}$ that validates the transactions contents and format against the current chain before including it in the transactions buffer. In existing systems such as Bitcoin, the $\mathsf{Validate}$ predicate checks the signature of the user spending funds. The function $\mathsf{Blockify}$, as in [KZZ16], handles the processing of the transaction buffer and "packs" it nicely into blocks.

Global Functionality The simulator for our protocol will not be able to act on behalf of $\mathcal{G}_{\mathsf{Ledger}}$. In particular the simulator is only able to use the functionality with the same priviliges as a party running the real protocol.

Chapter 3

Background and Technical Overview of Results

In this chapter, we describe the landscape of the state of the art for each problem that we discussed in the Introduction, and give a brief technical overview of how our work advances upon it.

3.1 Schnorr/EdDSA Signature Aggregation [CGKN21]

We view the problem of signature aggregation as that of constructing a compressing *proof of knowledge* of a collection of signatures. In particular, there must exist an extractor that outputs the constituent signatures with roughly the same probability that an adversary is able to produce an aggregate signature. To our knowledge, there does not exist such a construction (even for weaker game-based aggregate signatures [BGLS03]) for Schnorr or ECDSA signatures, besides the generic approach of expressing signature verification as an arithmetic circuit and proving knowledge of an input with a SNARK [BFH+20]. The current state of the art SNARKs [LSTW21, BFH+20] for this task require hundreds of milliseconds per signature that is aggregated, which as we described in the Introduction, induces far too high a latency for most applications.

Our Approach [CGKN21]. We start by constructing an n-special sound Sigma protocol to prove knowledge of n Schnorr signatures. Informally the Sigma protocol is the combination of the following two ideas:

1. Once m, pk, R are determined there is a unique $s \in \mathbb{Z}_q$ that 'completes' the signature, and this is the discrete logarithm of the publicly computable group element $S = H(\mathsf{pk}, R, m) \cdot$

$\mathsf{pk}+R$. Proving knowledge of the discrete logarithm of S is therefore equivalent to proving knowledge of the missing component of the signature. If m, pk are known, it therefore suffices to provide R and prove knowledge of $\mathsf{DLog}_G S$, in order to prove knowledge of the signature.

2. There is an n-special sound Sigma protocol to simultaneously prove knowledge of the discrete logarithms of n public group elements at the same bandwidth cost of a single PoK of DLog [GLSY04].

Combining these ideas yields the following n-special sound Sigma protocol: Upon fixing n messages m_i and signatures $(R_i, s_i)_{i\in[n]}$ under respective public keys pk_i, the prover is given a challenge $e \in \mathbb{Z}_q$, to which it computes the response $z = \sum_{i\in[n]} s_i \cdot e^i$. The verifier is given the statement $(\mathsf{pk}_i, R_i, m_i)_{i\in[n]}$, challenge e, and the putative Prover's response z, and validates them by verifying that $z \cdot G = \sum_{i\in[n]} e^i \cdot (H(\mathsf{pk}_i, R_i, m_i) \cdot \mathsf{pk} + R_i)$.

Theorem 3.1.1. *[CGKN21](Informal) There is an n-special sound Sigma protocol to prove knowledge of n Schnorr signatures, such that the size of the transcript of a single execution is half the size of a naive concatenation of those signatures.*

Applying the Fiat-Shamir transformation to this Sigma protocol yields a non-interactive proof of knowledge for n Schnorr signatures. In our work we demonstrated empirically [CGKN21, Table 1] that there is effectively no performance penalty incurred if one were to use this aggregation scheme rather than the naive concatenation method.

Caveat: Loose Security Proof. Our proof of this construction makes use of the Forking Lemma [PS00], which unfortunately results in a loss in (bits of) security proportional to the number of 'forks' required to extract the signature (at least two). In particular this means that in order to retain the same security level, the input signatures have to be doubled in size, which completely eliminates the 50% compression of the aggregation mechanism.

Achieving Provably Secure Parameters. We explore Fischlin's transformation [Fis05] as a method to obtain a non-interactive proof with a tight reduction via 'straight-line' extraction, while still retaining close to 50% compression. We found the computational cost of a naive application of Fischlin's transformation to be prohibitively high for many applications. We refer to this naive application of Fischlin's transformation when we cite [CGKN21] in the context of straight-line extraction henceforth. We postpone the exact figures to the next section, as in subsequent work [Ks22]—also contained in this thesis—we improve the cost of aggregation with straight-line extraction significantly, and provide a detailed comparison.

3.2 Improved Straight-Line Extraction [Ks22]

We first recall Fischlin's transformation in order to build intuition for our techniques. The base unit of the transformation is the following: for the instance x, the Prover computes a first message a of the Sigma protocol, and finds second and third messages e, z such that $V_x(a, e, z) = 1$ and $H(a, e, z) = 0$[1] for some ℓ-bit hash function H, where $\ell \in O(\log \kappa)$. This is done by starting with $e = 0$ (and the corresponding response z) and computing $H(a, e, z)$, iteratively stepping through e, z candidates which verify until the first e, z pair is found such that $H(a, e, z)$ evaluates to the all-zero string 0. An adversarial prover is able to produce (a, e, z) such that $H(a, e, z) = 0$ without querying more than one transcript to H only if it gets lucky with its first query, which happens with probability $2^{-\ell}$. This base unit is therefore repeated $r = \kappa/\ell$ times to achieve κ bits of soundness; specifically, to bind these instances together and prevent independent grinding, all of the a messages for the repeated instances are incorporated into the input to the hash function. For example, for $r = 2$ repetitions, the Prover must produce $a_1, a_2, e_1, e_2, z_2, z_2$ such that $H(a_1, a_2, e_1, z_1) = 0$ and $H(a_1, a_2, e_2, z_2) = 0$ and of course $V_x(a_1, e_1, z_1) = 1$ and $V_x(a_2, e_2, z_2) = 1$.

Prover Query Complexity. We refer to the (expected) number of queries that the prover makes to the random oracle as the *prover query complexity*. For instance, the Prover query complexity of Fischlin's construction as described above is $r \cdot 2^{\ell} = r \cdot 2^{\frac{\kappa}{r}}$, which implies a tradeoff between r (which governs proof size and verification cost) and the query complexity. We develop the study of prover query complexity in this work, as part of our study on the computation cost of straight-line extraction.

A note on exact vs. 'near' inversions. The version of the transformation described above is referred to as the 'basic' one by Fischlin. They proceed to tweak the Verifier to accept 'near' inversions, where it is sufficient for the Prover to output transcripts $\tau_1, \cdots, \tau_r$ such that $H(\tau_i)$ is interpreted as a positive integer and $\sum_i H(\tau_i) < S$ for some parameter $S \approx r$. The purpose of this change is to reduce the completeness error for the Prover (by increasing the soundness error). Our discussion on quasi-unique responses is unaffected by this change as the Prover is still deterministic and the same vulnerability persists. Regarding Prover query complexity, it is already pointed out in [Fis05] that relaxing this requirement for an accepting proof increases the soundness error, and adjusting the hash function parameter ℓ to retain the same r, κ values results in an increase in the expected Prover query complexity. Consequently we do not discuss the near-inversion variant further in this thesis, and every reference to Fischlin's construction will pertain to the basic exact inversion predicate.

[1] The instance x is also included in the hash, but omitted for clarity.

3.2.1 Schnorr/EdDSA Signature Aggregation and Computation Cost

Our motivating practical application is that of aggregating Schnorr/EdDSA signatures with tight security. As discussed previously, in [CGKN21] we construct a compressing Sigma protocol to prove knowledge of n Schnorr signatures, to which we apply Fischlin's transformation to obtain a non-interactive proof. Roughly, our initial 'baseline' scheme is to have the prover encode the n signatures as the coefficients of a degree $n-1$ polynomial f, and output a proof consisting of $(x_1, f(x_1)), \cdots, (x_r, f(x_r))$ such that each $H(x_i, f(x_i)) = 0$. We find producing such a proof to be computationally intensive, for instance over a minute to aggregate even hundreds of signatures at a 53% compression ratio[2] which induces a prohibitively high latency for many applications.

Faster Polynomial Evaluation with Curve25519. If we denote the prover query complexity as $\mathsf{T}_{\mathsf{Agg}}$, the prover must evaluate f at $\mathsf{T}_{\mathsf{Agg}}$ points. The first aspect of the prover's computation cost that we improve is the cost of producing $\mathsf{T}_{\mathsf{Agg}}$ evaluations of f. The naive method to evaluate a degree n polynomial costs n multiplications in $\mathbb{Z}_q$, meaning that the prover performs $n\mathsf{T}_{\mathsf{Agg}}$ multiplications. The Fast Fourier Transform (FFT) is a well-known method to speed up polynomial evaluation to $O(\mathsf{T}_{\mathsf{Agg}} \log n)$, and is used in straight-line extractable proofs for general statements [AHIV17, BCR+19]. Unfortunately the most common variant of Schnorr in practice—EdDSA—uses the elliptic curve group Curve25519 [Ber06], whose corresponding base field does not have a sufficiently large multiplicative subgroup to support the FFT. Asymptotically efficient polynomial evaluation algorithms for general fields [vzGG13, BCKL21] are not concretely efficient for the parameter range that is relevant to signature aggregation; we find our method to outperform the state of the art [BCKL21] by a factor of 5× even for the upper end of our signature aggregation parameters ($n = 1024$).

We make use of a method by which we can derive a randomly chosen polynomial h of degree $m < n$, such that it agrees with f on m points. Deriving h costs n multiplications, and evaluating h at each point costs m multiplications, which means that we can obtain m evaluations of f at roughly $n+m^2$ cost rather than the naive nm—a substantial improvement when $m \approx \sqrt{n}$. A prerequisite to use this method is that $\mathbb{Z}_q$ must have a multiplicative subgroup of size m, however unlike the FFT this method is *randomized* and can be invoked multiple times using the same subgroup, with negligible probability of producing redundant evaluations. Curve25519 has multiplicative subgroups of size up to 132, which provides nearly optimal values of $m \approx \sqrt{n}$ for the parameters relevant to signature aggregation.

[2] The r parameter governs a tradeoff between query complexity and compression ratio—a lower ratio is better compression, and 50% is the lowest possible [CGKN21]

Theorem 3.2.1. *[Ks22]Given a prime q, degree n polynomial $f \in \mathbb{Z}_q[X]$, and primitive k^{th} root of unity $\omega \in \mathbb{Z}_q$, there is an algorithm that outputs a list of k distinct points that lie on f at a cost of $k^2 + n + 2\log k$ multiplications and $k(k-1) + n$ additions in $\mathbb{Z}_q$.*

The intuition for the method is as follows: we decompose f into m different degree n/m polynomials f_i such that $f(x) = \sum_{i \in [m]} x^i \cdot f_i(x^m)$. We then sample $\alpha \leftarrow \mathbb{Z}_q$, and derive $h(x) = \sum_{i \in [m]} x^i \cdot f_i(\alpha^m)$. Observe that for any primitive m^{th} root of unity $\omega \in \mathbb{Z}_q$ and for any $j \in [m]$, it holds that $f_i((\alpha\omega^j)^m) = f_i(\alpha^m)$ for every f_i. Consequently, h agrees with f on the points $\{\alpha \cdot \omega^j\}_{j \in [m]}$.

Better Prover Query Complexity via Collisions. We change the underlying proof of work predicate to that of finding collisions rather than inversions of the hash function. In particular, the prover outputs a proof consisting of $(x_1, f(x_1)), \cdots, (x_r, f(x_r))$ such that $H(x_1, f(x_1)) = \cdots = H(x_r, f(x_r))$. For the same r and soundness level (note that ℓ has to be adjusted), analytical estimates on multicollision running times [vM39, Pre93] place the query complexity $\mathsf{T}_{\mathsf{Agg}}$ induced by this collision predicate at up to $2\times$ better than that of inversions.

Theorem 3.2.2. *(Informal) Let r be an integer, and H_1 and H_2 be random oracles with output lengths ℓ_1 and ℓ_2 bits respectively. Let* inv *and* col *be predicates such that* $\mathsf{inv}^{H_1}(x_1, \cdots, x_r) = 1$ iff $H_1(x_1) = \cdots = H_1(x_r) = 0^{\ell_1}$*, and* $\mathsf{col}^{H_2}(x_1, \cdots, x_r) = 1$ iff $H_2(x_1) = \cdots = H_2(x_r)$*. If r, ℓ_1, ℓ_2 are constrained so that* $\Pr[\mathsf{inv}^{H_1}(1, \cdots, r)] = \Pr[\mathsf{col}^{H_2}(1, \cdots, r)]$*, then finding a satisfying assignment for* col^{H_2} *is faster than finding one for* inv^{H_1}*.*

Combining these improvements (along with a tighter analysis that makes the proof of work easier by 2–8×) yields an improvement of a *factor of 70×–200×* for the most aggressive compression settings of the baseline approach reported in our prior work (see Table 3.2.3).

n	r	[CGKN21]		Improved [Ks22]		Improvement
		AggVer(ms)	AggSign	AggVer(ms)	AggSign	
512	16	137	167 ± 13.0 s	134	2.2 ± 0.07 s	76x
1024	32	485	85.5 ± 4.8 s	452 ± 6	350 ± 10 ms	244x
256	16	78	40.6 ± 2.8 s	72	901 ± 36 ms	45x
512	32	258	20.1 ± 1.4 s	255	136 ± 3 ms	147x
128	16	43	9.9 ± 0.74 s	42	363 ± 8 ms	27x
256	32	147	5.5 ± 0.31 s	143	54 ± 1 ms	101x
32	8	5.7	84.2 ± 11.6 s	5.6	7.8 ± 0.5s	11x
64	16	21	2.9 ± 0.25 s	23	78 ± 1 ms	37x
128	32	80	1.4 ± 0.08 s	84.5	20 ms	70x

Table 3.2.3: Comparing the computation cost for aggregation and aggregate-verification of n Ed25519 signatures with SHA-256 hash function used for H_1 on the same parameters from [CGKN21]. The benchmarks were run using the publicly available code for [CGKN21], and a new Rust implementation of our method and the Criterion rust framework; times show a 95% confidence interval over at least 30 runs on one Intel i7-10710U core running at 3.9Ghz with 32 Gb of memory. Intervals are omitted when less than 1ms. While the aggregation methods can easily be parallelized, each of these benchmarks only use 1-core to properly compare against the implementation from [CGKN21]. The best compression ratios are achieved on the first row at roughly 53%; the last row in the table achieves the worst ratio around 75%.

Collisions Improve Fischlin's NIZK. We generalize this principle and apply it to Fischlin's transform for NIZKPoKs as well, by using a collision pair base unit as a drop-in replacement for inversion base units. In particular, a collision pair base unit instructs the prover to find pairs of accepting Sigma protocol transcripts (a, e, z) and (a', e', z') such that $H((a, a'), e, z) = H((a, a'), e', z')$. A forgery requires a collision within the first two queries to the random oracle, which happens with probability $2^{-\ell}$ for an ℓ-bit hash function. This serves as a drop-in replacement for a pair of inversion base units that achieve a combined ℓ bits of soundness. Analyzing the query complexity is difficult as this is a *chosen prefix* collision [SLdW07], and so we test the new proof-of-work problem empirically and observe

an $11\% - 15\%$ improvement for common practical parameters. We report our results in Table 4.4.1 in Section 4.4.

A Query Complexity Lower Bound. We tighten Fischlin's asymptotic lower bound on hash queries for a NIZK with a non-programming extractor [Fis05, Proposition 2] to derive Lemma 4.4.2.

Lemma 3.2.4. *[Ks22](Informal)If a NIZKPoK scheme for a hard relation with a straight-line extractor (in the non-programmable ROM) induces a verifier to make V queries to the RO for a κ-bit security level, then the prover must on average make at least $P_{\mathsf{OPT}}[V, \kappa] = (V! \cdot 2^{\kappa})^{\frac{1}{V}}$ queries in generating a proof.*

Intuitively if the prover makes P queries of which V are checked by the verifier, $\binom{P}{V}$ must be at least 2^{κ} to achieve a $2^{-\kappa}$ soundness error. We note that this bound applies to schemes with perfect completeness, and while Lemma 4.4.2 is sufficiently general to derive a strict bound for probabilistic schemes, P_{OPT} serves as a useful reference point, and will be the quantity that we refer to as 'optimal' prover query complexity.

We show that the expected query complexity of Fischlin's construction is never better than $\sqrt{2}P_{\mathsf{OPT}}$ in any non-trivial parameter regime.

We note that Pass' transform (and equivalently Unruh's transform[3] [Unr15]) has a (strict) query complexity that is twice that of the expected prover complexity of Fischlin in any non-trivial parameter regime, and so we do not consider Pass/Unruh going forward.

Achieving P_{OPT}. For a special class of r-simulatable Sigma protocols (i.e. r transcripts are simulatable at once) we show that a NIZKPoK with prover query complexity P_{OPT} can be achieved for a range of non-trivial parameters. We construct this NIZK by applying a multicollision predicate akin to our signature aggregation construction, where the prover must produce transcripts $(\boldsymbol{a}, e_1, z_1), \cdots, (\boldsymbol{a}, e_r, z_r)$ such that $H(\boldsymbol{a}, e_1, z_1) = \cdots = H(\boldsymbol{a}, e_r, z_r)$. We make use of classic results on multicollision complexities [vM39, Pre93] to analyze the expected prover query complexities, which we denote $\mathsf{T}_{\mathsf{Col}}$.

Lemma 3.2.5. *[Ks22](Informal) There is a NIZPoK for the DLog relation with a straight-line extractor (in the non-programmable ROM) where the prover makes roughly $P_{\mathsf{OPT}}[V, \kappa]$ queries on average for V up to 5, and $\kappa = 128$ onwards.*

[3]For the purpose of prover query complexity, Unruh's transform can be seen as Pass' transform without the Merkle trees to reduce the number of repetitions of the base Sigma protocol.

Note that this transform is limited in applicability—we show how Schnorr's proof of knowledge of discrete logarithm can be made r-simulatable, but leave it as an interesting problem for future work to expand the scope of this transform.

Wider Application of Our Techniques. Our techniques for improving the computation cost of Signature Aggregation can be applied directly to the threshold cryptography context for the same signature schemes. For example, the most expensive component of Distributed Key Generation (DKG) for the canonical (t, n) threshold Schnorr scheme [Lin22, Protocol 6.1] is the NIZKPoK to prove knowledge of a polynomial that is committed in the curve group. The instantiation for this NIZKPoK suggested by Lindell [Lin22] is the batch PoK of Discrete Log [GLSY04] compiled to a NIZK using Fischlin's transform—i.e. exactly the same as EdDSA signature aggregation (with an extra blinding factor). Consequently, DKG for (t, n) EdDSA can benefit from roughly the same speedup that we report for signature aggregation.

3.2.2 Extending the Applicability of Fischlin's Transform

A technicality in Fischlin's transformation arises when it is possible for the Prover to iterate through verifying transcripts *without* having to change the challenge message e. Consider a Sigma protocol that permits an adversary without a witness to sample $(a, e), z_1, z_2, \cdots z_n$ such that each (a, e, z_i) is a valid transcript. Applying Fischlin's transformation will not produce a sound NIZK because an adversary can simply step through $H(a, e, z_1)$, $\cdots$, $H(a, e, z_n)$ to find a pre-image of 0 whereas an extractor may not be able to extract a witness from this sequence of queries because they do not satisfy the requirements for 2-special soundness.

Although it is folklore that many Sigma protocols allow for extraction even given accepting transcripts $(a, e, z_1), (a, e, z_2)$ (examples include the famous logical OR composition [CDS94], opening of a Pedersen commitment, etc. for which this is simply a matter of adjusting syntax), Fischlin's transform only applies to protocols that support a *quasi-unique response* property, given below.

Definition 3.2.6. *[Fis05, Definition 1] A Sigma protocol has quasi-unique responses if for every PPT algorithm $\mathcal{A}$, for system parameter k and $(x, a, e, z_1, z_2) \leftarrow \mathcal{A}(k)$, we have as a function of k that the following probability is negligible:*

$$\Pr\left[V_x(a, e, z_1) = V_x(a, e, z_2) = 1 \wedge z_1 \neq z_2\right]$$

Here the system parameter k can be an arbitrarily structured object sampled according to some distribution, for eg. an RSA modulus or $h \in \mathbb{G}$ such that $\mathsf{DLog}_g(h)$ is unknown, as required in Okamoto's identification protocols [Oka93].

Interestingly, Fischlin's proof also uses this property to argue *zero-knowledge*. It is less obvious as to why quasi-unique responses is relevant for this purpose. In the absence of an explicit attack on the zero-knowledge property when quasi-unique responses does not hold, one may even conclude that it is simply an artefact leveraged to prove the simulation secure.

We show this intuition to be *false*. In particular, we construct an explicit attack on *Witness Indistinguishability* when Fischlin's transformation is applied to a common Sigma protocol for a language with two witnesses. This attack is the result of combining two facts:

- **Fischlin's Transformation is Deterministic.** Once the Sigma protocol first messages have been sampled, the prover's algorithm is deterministic.
- **Some Sigma Protocols Reveal the Prover's Randomness.** In particular Schnorr's proof of knowledge of discrete logarithm reveals a linear combination of the witness and the prover's randomness—knowledge of the witness therefore allows an attacker to reconstruct the prover's randomness.

It is therefore possible for an attacker to *retrieve* the prover's random tape when given a Fischlin-compiled Schnorr proof, and *replay* the prover's steps and reconstruct the proof string. To demonstrate why this is problematic, we examine the effect of this retrieve-and-replay strategy given a Fischlin-compiled proof of knowledge of one-out-of-two discrete logarithms [CDS94]. In particular if a prover uses one of x_0, x_1 to prove knowledge of $x_0 \cdot G \vee x_1 \cdot G$, an attacker with knowledge of say x_0 can execute the retrieve-and-replay strategy to test if x_0 was indeed used in producing the proof string. We show that if the attacker uses x_0 to execute this strategy on a proof that was actually produced using x_1, there is a non-negligible chance that the proof string that the attacker reconstructs will be *different* from the given one (as opposed to a proof string produced using x_0, which always matches the reconstruction). Intuitively, this is because the proof string serves as a record of how many Sigma protocol transcripts had to be hashed before a solution to the proof of work was found—recomputing the proof using a different witness might result in finding a solution by hashing fewer transcripts.

Theorem 3.2.7. *[Ks22](Informal) Fischlin's transformation does not preserve Witness Indistinguishability when applied to the Sigma protocol to prove knowledge of one of two Discrete Logarithms.*

We note that our attack runs entirely in the random oracle model and does not exploit concrete instantiations of the hash function, unlike previous work that studies the concrete instantiability of Fischlin's transform [ABGR13].

Randomization Fixes the Problem. We formalize a notion of *strong special soundness* to capture the folklore notion that accepting transcripts of the form (a, e, z_1),(a, e, z_2) yield a witness. This is a subtle change in the definition of special soundness; luckily many natural Sigma protocols (including those with multiple witnesses for which Fischlin's transformation is shown not to work as above) satisfy this property, including every regular special sound Sigma protocol that supports quasi-unique responses.

We then show how to randomize Fischlin's transformation to erase all traces of the witness from the compiled proof strings, and prove that zero-knowledge is guaranteed unconditionally for any strong special sound Sigma protocol. Intuitively this is achieved by having the prover step randomly through the challenge space to find a solution to the proof of work, and this form of randomization is directly compatible with a collision-based proof of work.

Theorem 3.2.8. *[Ks22](Informal) Any Strong Special Sound Sigma protocol can be compiled to a straight-line extractable NIZKPoK in the ROM, with the same computation and bandwidth efficiency as applying Fischlin's transformation.*

Our attack on WI appears to uncover an interesting aspect of the role of randomness in straight-line extractable zero-knowledge proofs. Pass' transformation is randomized (due to its use of a commitment scheme), and naively derandomizing it would result in a similar attack. An interesting and natural question for future work would be to identify the class of languages for which "well-behaved" transforms that make black-box use of an underlying zero-knowledge protocol and compile them into a straightline extractable one in the random oracle model *must* be randomized.

We therefore demonstrate conclusively that one can do better than generic cut-and-choose (i.e. Pass [Pas03]) for straight-line extractable NIZKs for many algebraic languages in the random oracle model. Such languages include logical combinations [CDS94], openings to Pedersen commitments, among many others that are used in non-trivial cryptographic systems such as the anonymous survey protocol [HMPs14].

3.3 Stateless Deterministic Threshold Schnorr Signing [GKMN21]

A naive application of the technique used by EdDSA to the multiparty case (i.e. each party derives its respective nonce as $k_i = \mathsf{F}(\mathsf{sd}_i, m)$) yields stateless deterministic threshold signing for Schnorr in the semi-honest setting. However Maxwell et al. [MPSW19] identified that this technique is completely insecure in the malicious setting, due to a 'rewinding' attack where

an adversary can effectively force a nonce reuse by an honest party. Intuitively, the adversary initiates two independent signing sessions where the same message is to be signed, but uses two different values as its contribution to the nonce, which induces the honest party to reveal two different linear combinations of the same secret key and nonce shares. We revisit this issue in Section 5.1.1

As Nick et al. [NRSW20] identified, a conceptually simple fix for this problem is to have each party commit to a PRF key sd during key generation, and subsequently prove when signing a message that its contribution to the nonce R is indeed the result of using its PRF key. Informally, the task is to design a zero-knowledge proof system for the following language:

$$\{((R, m, F, \mathbb{G}, C), \mathsf{sd}) : R = \mathsf{F}(\mathsf{sd}, m) \cdot G \ \wedge \ C = \mathsf{Commit}(s)\}$$

where sd is the witness to be kept hidden. More formally, we encapsulate this task in Functionality 5.6.1 in Section 5.6.

While there is a plethora of proof systems one could use to prove the above relation, as our goal is to design a stateless deterministic threshold signing protocol that is faster than using trusted hardware, we opt for the computationally lightest approach. In particular, we follow the zero-knowledge from garbled circuits (ZKGC) paradigm of Jawurek et al. [JKO13] along with a conditional disclosure technique [GKPS18]. We defer a more thorough explanation as to why this paradigm suits the task at hand best to Section 5.1.3.

The difficulty in proving the above relation lies in combining the algebraic component (i.e. elliptic curve exponentiation) with the non-algebraic component (F instantiated with AES/SHA). In this ZKGC context, the heaviest elements lie in (1) the bridge between the algebraic and non-algebraic components, and (2) public key operations required for committing OT. We develop new techniques to suppress both, so that they are no longer bottlenecks.

3.3.1 Bridging Algebraic and Non-algebraic Operations

While previous work by Chase et al. [CGM16] has explored this topic in the ZKGC context, their solution makes use of far too many public key operations to be computationally efficient. We give a new garbling gadget for the exponentiation function that significantly improves on their work, intuitively by making use of the Oblivious Linear Evaluation technique of Gilboa [Gil99] to apply an algebraic MAC on the non-algebraic component.

Intuition. The ciphertexts are structured so that the evaluator always decrypts $z_i = b_i + \mathbf{x}_i \cdot \mathbf{u}_i \cdot a$ on wire i, where a and $b = \sum_i b_i$ are the garbler's MAC keys and $x = \langle \mathbf{u}, \mathbf{x} \rangle$. Adding up $z = \sum_i z_i$ yields $z = ax + b$, which is the desired arithmetic encoding, and allows for easy

exponentiation outside the garbled circuit. This self-contained gadget can be expressed as a garbling scheme and proven secure as such. We give the concrete improvement relevant for our parameters in Table 3.3.1.

Scheme	Asymptotic Comm.	Concrete Comm.	Calls to KDF
[CGM16]	$\tilde{O}(s \cdot \lvert q \rvert \cdot \kappa)$	1024KB	64000
Our gadget	$O(\lvert q \rvert \cdot \kappa)$	8.2KB	1024

Table 3.3.1: Cost to apply algebraic MAC $z = ax + b$ to a secret x encoded in a garbled circuit. Concrete costs are given for $|q| = 256$, $s = 60$, and $\kappa = 128$, with the HalfGates [ZRE15] garbling scheme. KDF is the cipher used for garbling.

Our technique leads to significant savings, as stated earlier the MAC computation alone would have dominated bandwidth (and to some extent computation) cost.

3.3.2 Committed Oblivious Transfer

The ZKGC protocol makes use of 'committed' OT, which is a flavour of OT in which a sender is able to decommit the messages it had sent earlier, upon request. Unfortunately all known efficient constructions require public key operations per instance, which can be quite heavy in this context, as they will induce hundreds of exponentiations per signing instance. We therefore design a committed OT mechanism that pushes all of the public key operations to the one-time distributed key generation.

Intuition. Recall that a UC commitment scheme must be 'straight-line extractable', i.e. there must exist an extractor algorithm Ext, which when given a commitment C to message m and a trapdoor ek should efficiently output m. Our insight is to run Ext to implement the committed OT receiver, even though its utility in the context of the UC commitment is simply as a 'proof artefact' which is never executed in a real protocol. Roughly, we generate a pair of commitment keys $\mathsf{ck}_0, \mathsf{ck}_1$ for the OT sender during the preprocessing phase, and give the trapdoor ek_b corresponding to ck_b to the OT receiver, where b is the choice bit. To send a message pair (m_0, m_1) the sender commits to m_0 using ck_0 and m_1 using ck_1, of which the receiver retrieves m_b by invoking Ext with ek_b. In order to 'open' its messages, the sender simply runs the decommitment phase of the UC commitment scheme. The novelty in this approach lies in our use of the extraction trapdoor ek, which is an object that only appears in the security proof of a UC commitment (but not in the 'real' world), to construct a concrete

protocol. The real-world OT receiver essentially runs the simulator of the UC commitment scheme.

We give an estimate below of the computation cost per OT instance, where runtime figures are taken from the author's 2017 Macbook Pro (i7-7700HQ CPU, 2.80GHz) running OpenSSL 1.1.1f.

Scheme	Comp.	Comm. (bits)	Estd. runtime
OT [CO15]	5 exponentiations	1152	$299\mu s$
This work	$240 \cdot \mathsf{F} + 31 \cdot \mathsf{CRHF}$	5120	$26.1\mu s$

Table 3.3.2: Cost per bit of the witness (send+receive+open), per instance not including preprocessing. Parameters: 128 bits of computational security, 60 bits of statistical security. Estimated runtime with AES for F, SHA-512 for CRHF, and Curve25519 for exponentiations.

With the above improvements, we obtain a proof system that induces only a small constant number of exponentiations per threshold signing instance.

Theorem 3.3.3. *[GKMN21](Informal) Assuming the existence of privacy-free garbled circuits [FNO15, ZRE15] and collision resistant hash functions, there is a protocol to UC-realize $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid local random oracle model, where each invocation of the 'Verify Nonce' interface induces $O(1)$ exponentiations and $O(\kappa|\mathsf{F}|)$ bits of communication.*

We estimate that a two-party signing instance for standard 256-bit curves will run in the order of 10ms on commodity hardware (which is signficantly faster than trusted hardware, which takes high tens to hundreds of milliseconds [SP16, BCLK17]), while consuming around 400KB of bandwidth. In contrast, the construction of Nick et al. [NRSW20] requires only a few hundred bytes of bandwidth, but 1 second to produce a proof, when instantiated with Bulletproofs [BBB+18] and a custom elliptic curve based PRF. We refer the reader to Chapter 5 for details about the setting, constructions, and comparisons.

3.4 Proactive Threshold Wallets With Offline Devices [KMOS21]

Proactive Secret Sharing (PSS) as it has come to be known, has seen a number of realizations for different ranges of parameters since the introduction of the mobile adversary model [OY91]. In fact, even proactive signature schemes themselves have been studied directly [ADN06, FGMY97]. A naive adaptation of any off-the-shelf PSS scheme to the thresh-

old signature setting would in many cases yield proactive threshold signature schemes immediately. However, heavy use of an honest majority by most PSS schemes would already rule out many practical applications of such an approach. Moreover all such solutions will have communication patterns that require every party in the system to be online at pre-defined times, at the close of *every* epoch, in order to keep the system proactivized and moving forward.

To see why requiring all parties to be online simultaneously is not reasonable especially for threshold wallets, consider the following scenarios:

- **Cold storage:** Alice splits her signing key between her smartphone and laptop and has them execute a threshold signing protocol when a message is to be signed. However if for any number of operational reasons one of the devices (say her smartphone) malfunctions, the secret key is lost forever and any funds associated with the corresponding public key are rendered inaccessible. In order to avoid this situation, Alice stores a third share of the signing key in a secure *cold storage* server. While this third share does not by itself leak the signing key, along with the laptop it can aid in the restoration of the smartphone's key share when required. In this scenario it would be quite inconvenient (and also defeat the purpose of two-party signing) if the cold storage server has to participate in the proactivization every time the system needs to be re-randomized; it would be much more reasonable to have the smartphone and laptop proactivize when required, and send update packages to the server.

- **(2,3)-factor authentication:** Alice now splits her signing key across her smartphone, laptop, and tablet so that she must use any two of them to sign a message. Even in this simple use case, having all of her devices online and active simultaneously (possibly multiple times a day) just so that they can refresh would be cumbersome. Ideally every time she uses two of them to sign a message, they also refresh their key shares and leave an update package for the offline device to catch up at its leisure.

- **Concurrent use:** Alice, Bob, Carol, and Dave are executives at a corporation, and at least two of them must approve a purchase funded by the company account. This is enforced by giving each of them a share of the signing key, so that any two may collaborate to approve a transaction. Requiring them all to be online simultaneously is impractical given their schedules; it would be much more convenient to have any two of them refresh the system when they meet to sign, and send updates to the others.

Correlated Risks Beyond convenience, there are qualitative security implications for the de-facto standard pattern of proactivization. In particular, the validity of the assumption

that an adversary controls only up to a threshold number of devices hinges on the risk of compromise of each device being independent. However having all devices in the system come online at frequent pre-specified points in time and connect to each other to refresh may significantly correlate their risk of compromise. Instead it would be preferable that only the minimal number of devices (i.e. the signing threshold) interact with each other in the regular mode of operation, and enable the system to non-interactively refresh itself.

The ideal communication pattern alluded to above is the following: in a (t, n) proactive threshold signature scheme, any t parties are able to jointly produce all the necessary components to refresh the system, and send the relevant information to offline parties. When an offline party wakes up, it processes the messages received and is able to "catch up" to the latest sharing of the secret.

3.4.1 Challenges in Realizing this Pattern

While this communication pattern sounds ideal, a whole host of subtle issues arise in potential realizations. For instance, in the Cold Storage case, how does the server know that the updates it receives are "legitimate"? An attacker controlling Alice's smartphone could spoof an update message and trick the server into deleting its key share and replacing it with junk.

Due to the inherent unfairness of two-party/dishonest majority MPC protocols, an adversary can obtain the output of the computation while depriving honest parties of it. In this spirit, the smartphone (acting for the attacker) could work with the laptop until it obtains the "update" message to send to the server, but abort the computation before the laptop gets it. Now the attacker has the ability to convince the server to delete its old share by using this message, whereas the laptop has no idea whether the attacker will actually do this (and therefore doesn't know whether to replace its own key share).

Implicit in these scenarios is the problem of *unanimous erasure*:

> How can we design a proactivization protocol in which the adversary can not convince an honest party to prematurely erase its secret key share?

In the $(2, 2)$ case even a network adversary (who does not control either party) can induce premature deletion by simply dropping a message in the protocol. Moreover is it possible to restrain such a proactivization procedure to be *minimally invasive* to the threshold wallet? i.e. native to usage patterns and protocol structures of threshold wallets.

3.4.2 Our Contributions

In this work we give a comprehensive treatment of the notion of proactive security with offline-refresh, with our study progressing in four phases:

1. **Defining Offline Refresh.** We formalize the notion of offline refresh for threshold protocols in the Universal Composability (UC) framework [Can01], and justify why our definition (unanimous erasure) is the correct one. Our starting point is the definition of Almansa et al. [ADN06] which we build on to capture that all parties need not be in agreement about which epoch they are in, and that an adversary can change corruptions while other parties are offline. Intuitively previous definitions have had an inherent synchrony in the progress of the system, which we remove in ours and show how to capture that parties may refresh at different rates.

2. **Upgrading $(2, n)$ Schemes.** We show how to upgrade $(2, n)$ threshold Schnorr-like signature schemes to proactive security tailored for use with a threshold wallet, in that it makes use of transactions posted to the blockchain for synchronization purposes. We make the case in Section 6.2.2 that the power of a ledger is necessary for this task. Our refresh protocol adds no extra assumptions, incurs very little overhead as compared to running the threshold signature itself, and exactly matches the *ideal* communication pattern outlined in the previous section.

3. **Proactive Multiplication.** We construct a mechanism to proactivize OT Extension state. This allows us to proactivize even threshold ECDSA protocols, which are sophisticated due to the non-linear signing equation. We prove the efficiency of our construction by means of an implementation, specifically the overhead incurred in computational time of our refresh procedure is roughly 24% for the ECDSA protocol of Doerner et al. [DKLs19] and the communication round overhead is zero.

4. **Impossibility of Online Dishonest Majority for $(3, n)$ and Beyond.** Intuition would strongly suggest that any (t, n) threshold scheme could also be upgraded to proactive security with offline refresh in the presence of a dishonest majority online using sufficiently heavy cryptographic hammers. However, surprisingly we show this intuition to be false; i.e. even assuming arbitrary trusted setup/random oracle and an ideal ledger, *there must be an honest majority online* to refresh the system. We prove this result by developing new elegant techniques to reason about security in this setting.

We therefore formulate the problem of offline refresh and address the most pressing practical and theoretical questions: the honest majority online case is simple, the $(2, n)$ case permits a

novel efficient protocol with a ledger which we implement, and the (t, n) case for $t > 2$ must necessarily have an honest majority of participants online.

Broader Implications Our results can be interpreted as positive for small-scale decentralization, eg. 2FA across personal devices. In particular the $(2, n)$ refresh protocol is readily compatible with existing implementations of threshold wallets, and essentially comes at only the cost of implementing forward-secure channels. However our impossibility result rules out this strong form of security for larger scale systems, where many servers hold shares of a secret with a high reconstruction threshold. In those cases system designers who desire proactive security must account for the cost of either bringing an honest majority online, or waiting to hear from all parties before progressing epochs.

3.4.3 Our Techniques

We first sketch the ideas behind our $(2, n)$ construction, and then discuss how to reason about the general (t, n) case and show impossibility.

$(2, n)$ Construction

Roughly, our approach is to use private channels to communicate candidate refresh packages, and the public ledger to achieve consensus on which one to use. We take advantage of the fact that threshold wallets already rely on posting signatures to a public ledger in order to coordinate these refreshes. Let each party P_i own point $f(i)$ on a shared polynomial f where $f(0) = \mathsf{sk}$ (i.e. standard Shamir sharing of the secret key sk). We have parties generate a candidate refresh polynomial f' when they sign a message, associate each signature with f', and "apply" the refresh (i.e. replace $f(i)$ with $f'(i)$) when the corresponding signature appears on the blockchain. While this handles the coordination part, the major issue of verifiably communicating $f'(j)$ to offline party P_j remains a challenge. To solve this, we have the online refreshing parties jointly generate a *local* threshold signature authenticating f' when communicated to each offline party; such a signature can only be produced by two parties working together, so any candidate f' received when offline must have been created with the approval of an honest party.

Working Around Unfairness Note that this approach is still vulnerable to attacks where the adversary withholds the threshold signature from an honest party in the protocol; if an online signing protocol aborts, how does an honest party know if its (possibly malicious) signing counterparty sent f' and the corresponding signature to offline parties? This is an issue

that stems from the inherent unfairness of two-party computation. While this is impossible to solve in general, we observe that most threshold ECDSA/Schnorr signature protocols are simulatable so the signing nonce R is leaked, but the signature itself stays hidden until the final round. We exploit this fact to bind each f' to R instead of the signature itself; so our proactive version of threshold ECDSA/Schnorr will proceed as follows:

1. Run the first half of threshold ECDSA/Schnorr to obtain R.
2. Sample candidate f', bind it to R, threshold-sign these values and send them to offline parties.
3. Continue with threshold ECDSA/Schnorr to produce the signature itself.

Correspondingly when *any* signature under R appears on the blockchain, each party searches for a bound f' that it can apply. With overwhelming probability there will never be two independently generated signatures that share the same R nonce throughout the lifetime of the system.

Threshold ECDSA and Multipliers Threshold ECDSA protocols require use of a secure two-party multiplication functionality $\mathcal{F}_{\mathsf{MUL}}$ (or equivalent protocol) due to its non-linear signing equation. Indeed, recent works [GG18, LNR18, DKLs19] have constructed practical threshold ECDSA protocols that make use of multipliers that can be instantiated with either Oblivious Transfer or Paillier encryption. Using these multipliers is significantly more efficient in the offline-online model where parties run some kind of preprocessing in parallel with key generation, and make use of this preprocessed state for efficient $\mathcal{F}_{\mathsf{MUL}}$ invocation when signing a message (this is done by all cited works). However as this preprocessed state is persistent across $\mathcal{F}_{\mathsf{MUL}}$ invocations, it becomes an additional target to defend from a mobile adversary. We show how to efficiently re-randomize this preprocessed state for OT-based instantiations of $\mathcal{F}_{\mathsf{MUL}}$, and therefore get offline-refresh proactive security for $(2, n)$ threshold ECDSA in its entirety. Our proactivization of $\mathcal{F}_{\mathsf{MUL}}$ makes novel use of the classic technique of Beaver [Bea95] to preprocess oblivious transfer, in combination with the mechanism we build to deliver updates securely.

General (t, n) Impossibility

We develop a novel technique to reason about the security of protocols that tolerate mobile corruptions. We first prove that any refresh protocol that tolerates an online dishonest majority must have the property that a minority of online parties holds enough information to allow any offline party to refresh. Subsequently we show that a mobile adversary can exploit

this property to derive the refreshed private state of a previously corrupt offline party even after it is un-corrupted. The proof is built up from this underlying insight, discussed further in Section 6.8.

3.4.4 Related Work

The notion of mobile adversaries with a corresponding realization of proactive MPC was first introduced by Ostrovsky and Yung [OY91]. Herzberg et al. [HJKY95] devise techniques for proactive secret sharing, subsequently adapted for use in proactive signature schemes by Herzberg et al. [HJJ+97]. Cachin et al. [CKLS02] show how to achieve proactive security for a shared secret over an asynchronous network. Maram et al. [MZW+19] construct a proactive secret sharing scheme that supports dynamic committees, with a portion of the communication done through a blockchain. For a more comprehensive survey, we refer the reader to the works of Maram et al. [MZW+19] and Nikov and Nikova [NN05].

Very recently Benhamouda et al. [BGG+20] and Goyal et al. [GKM+20] introduced a protocol in which a committee (elected from a larger set of parties) runs what is essentially a proactivization with offline-refresh. However they work in the setting of an honest majority, and their techniques are tailored as such.

The work of Canetti et. al. [CHH00] solves the problem of an offline node regaining the ability to authenticate its communication after having suffered a break-in. However the settings are incomparable; our network model is stronger in that we assume authenticated communication (details in Section 6.2), but weaker in another dimension as we do not rely on an honest majority among online parties. Our use of the ledger is merely as a passive public signalling mechanism, and not as interactive party-specific storage (eg. no issuing of certificates to individual parties).

As discussed earlier, *every* existing work (including those since the above mentioned surveys) assumes either that all parties come online [CGG+20], an honest majority of parties collaborate in order to proactivize the system [BGG+20, GKM+20], or that corruptions are passive [EOPY18]. Additionally they require this honest majority of parties to come online simultaneously at pre-specified points in time to run the refresh protocol. As the entire premise of the (t, n) threshold signature setting is that only t parties need be online simultaneously to use the system,

- For the $(2, n)$ case we impose as a strict requirement that only two parties be sufficient to proactivize the system. Consequently as it is meaningless to have an honest majority among two parties, we can not directly apply techniques from previous works to our setting. To our knowledge the conceptual core of our protocol– a threshold signature (internal to

the system) interleaved with a threshold signature that appears on the blockchain, is novel.

- For the general $t > 2$ case, we prove that the weakest possible notion of dishonest majority for proactivization, i.e. refresh with $2t - 1$ online parties, is impossible to achieve.

Therefore we give a comprehensive treatment of proactivization with an online dishonest majority, which has not previously been studied in the literature.

Chapter 4

Schnorr Signature Aggregation and Improved Straight-Line Extraction in the Random Oracle Model

In this chapter, we present our results on signature aggregation, and straight-line extraction in the random oracle model.

4.1 Signature Aggregation

We first explore aggregating EdDSA signatures as a motivating practical application. We begin by giving our n-special sound Sigma protocol to prove knowledge of n Schnorr signatures. Define the relation R_{Agg} as:

$$R_{\mathsf{Agg}} = \{(x, w) \mid x = (\mathsf{pk}_1, m_1, \ldots, \mathsf{pk}_n, m_n), w = (s_1, \ldots, s_n), \\ \mathsf{Verify}(m_i, \mathsf{pk}_i, s_i) = \mathsf{true} \text{ for } \forall i \in [n]\}$$

i.e. each $s_i \in \mathbb{Z}_q$ is a signature on message $m_i \in \{0,1\}^*$ under Schnorr public key $\mathsf{pk}_i \in \mathbb{G}$, as per the Schnorr Verify algorithm.

Theorem 4.1.1. *Protocol Σ_{aggr} is an n-special sound Sigma protocol for R_{Agg}.*

Proof. Completeness is easy to verify. Extraction is always successful due to the following: let $F \in \mathbb{G}[X]$ be the degree $n-1$ polynomial where the coefficient of x^{i-1} is given by $R_i + H(R_i, \mathsf{pk}_i, m_i) \cdot \mathsf{pk}_i$ for each $i \in [n]$. Define $f \in \mathbb{Z}_q[X]$ as the isomorphic degree $n-1$ polynomial over $\mathbb{Z}_q$ such that the coefficient of x^{i-1} in f is S_i (the discrete logarithm of the corresponding coefficient in F). Observe that $f(x) \cdot B = F(x)$ for each $x \in \mathbb{Z}_q$. Given

Protocol Σ_{aggr}

For instance $x = \{(\mathsf{pk}_i, m_i)\}_{i=1}^n \in (\mathbb{G} \times \{0,1\}^*)^n$ and witness $w = \{\sigma_i = (R_i, s_i)\}_{i=1}^n \in (\mathbb{G} \times \mathbb{Z}_q)^n$

Prover $P_\Sigma(x, w)$:

1. **Commitment**: $a = [R_1, \ldots, R_n]$
2. **Challenge**: $e \xleftarrow{\$} \mathbb{Z}_q^*$
3. **Response**: $z = \sum_{i \in [n]} s_i \cdot e^{i-1}$

Verifier $V_\Sigma(x, (a, e, z))$: Output 1 iff $z \cdot B = \sum_{i \in [n]} e^{i-1}(R_i + H(R_i, \mathsf{pk}_i, m_i) \cdot \mathsf{pk}_i)$

Extractor $\mathsf{Ext}_\Sigma((a, e_1, z_1), \ldots, (a, e_n, z_n))$: Define the $n \times n$ matrix $E = [e_i^j]_{i,j \in [n]}$ and the column vector $Z = ([z_i]_{i \in [n]})^T$. Output $(s_1, \ldots, s_n) = (E^{-1}Z)^T$.

Figure 4.1: Sigma protocol for a collection of signatures R_{Agg}

a transcript (a, e, z), V_Σ accepts iff $z \cdot B = F(e)$, which is true iff $z = f(e)$. Therefore n valid transcripts $(a, e_1, z_1), \ldots, (a, e_n, z_n)$ define n distinct evaluations of f (which has degree $n-1$) allowing for recovery of coefficients $[S_i]_{i \in [n]}$ efficiently. This is precisely the operation carried out by Ext_Σ, expressed as a product of matrices. Note that $E = [e_i^j]_{i,j \in [n]}$ is always invertible; each e_i is known to be distinct, and so E is always a Vandermonde matrix. □

For signatures instantiated over a field of order q, the transcript of the Sigma protocol is of size $(n+1)|q|$ bits, as opposed to naive transmission of n signatures which would require $2n|q|$ bits. It is straightforward to apply the Fiat-Shamir transformation to this Sigma protocol and obtain a non-interactive proof of knowledge for the relation R_{Agg}, and so we refer the reader to the full version of the paper [CGKN21] for a thorough treatment.

We do however justify that this is the best possible compression rate that one can achieve with techniques that are blackbox in the hash function used by Schnorr.

4.2 Impossibility of non-interactive compression by more than a half

Given that we have shown that it is possible to compress Schnorr signatures by a constant factor, it is natural to ask if we can do better. Indeed, the existence of succinct proof systems where the proofs are smaller than the witnesses themselves indicates that this is possible, even

without extra assumptions or trusted setup if one were to use Bulletproofs [BBB+18] or IOP based proofs [BBHR18, BCR+19] for instance. This rules out proving any non-trivial lower bound on the communication complexity of aggregating Schnorr's signatures. However, one may wonder what overhead is incurred in using such generic SNARKs, given their excellent compression. Here we make progress towards answering this question, in particular we show that non-trivially improving on our aggregation scheme must rely on the hash function used in the instantiation of Schnorr's signature scheme.

We show in Theorem 4.2.1 that if the hash function used by Schnorr's signature scheme is modeled as a random oracle, then the verifier must query the nonces associated with each of the signatures to the random oracle. Given that each nonce has 2κ bits of entropy, it is unlikely that an aggregate signature non-trivially smaller than $2n\kappa$ can reliably induce the verifier to query all n nonces.

The implication is that an aggregation scheme that transmits fewer than $2n\kappa$ bits must not be making oracle use of the hash function; in particular it depends on the code of the hash function used to instantiate Schnorr's scheme. To our knowledge, there are no hash functions that are believed to securely instantiate Schnorr's signature scheme while simultaneously allowing for succinct proofs better than applying generic SNARKs to their circuit representations. Note that the hash function must have powerful properties in order for Schnorr's scheme to be proven secure, either believed to be instantiating a random oracle [PS00] or having strong concrete hardness [NSW09]. Given that the only known techniques for making use of the code of the hash function in this context is by using SNARKs generically, we take this to be an indication that compressing Schnorr signatures with a rate better than 50% will incur the overhead of proving statements about complex hash functions. For instance compressing n Ed25519 signatures at a rate better than 50% may require proving n instances of SHA-512 via SNARKs.

For "self-verifying" objects such as signatures (aggregate or otherwise) one can generically achieve some notion of compression by simply omitting $O(\log \kappa)$ bits of the signature string, and have the verifier try all possible assignments of these omitted bits along with the transmitted string, and accept if any of them verify. Conversely, one may instruct the signer to generate a signature such that the trailing $O(\log \kappa)$ bits are always zero (similarly to blockchain mining) and need not be transmitted (this is achieved by repeatedly signing with different random tapes). There are two avenues to apply these optimizations:

1. **Aggregating optimized Schnorr signatures.** One could apply these optimizations to the underlying Schnorr signature itself, so that aggregating them even with our scheme produces an aggregate signature of size $2n(\kappa - O(\log \kappa))$ which in practice is considerably better than $2n\kappa$ as n scales. In the rest of this section we only consider the aggregation of

Schnorr signatures that are produced by the regular unoptimized signing algorithm, i.e. where nonces have the full $2n\kappa$ bits of entropy. This quantifies the baseline for the most common use case, and has the benefit of a simpler proof. However, it is simple to adapt our proof technique to show that aggregation with compression rate non-trivially greater than 50% is infeasible with this optimized Schnorr as the baseline as well.

2. **Aggregating unoptimized Schnorr signatures**. One could apply this optimization to save $O(\log \kappa)$ bits overall in the aggregated signature. In this case, $O(\log \kappa)$ is an additive term in the aggregated signature size and its effect disappears as n increases, and so we categorize this a trivial improvement.

Proof Intuition. Our argument hinges on the fact that the verifier of a Fiat-Shamir transformed proof must query the random oracle on the 'first message' of the underlying sigma protocol. In Schnorr's signature scheme, this represents that the nonce R must be queried by the verifier to the random oracle. It then follows that omitting this R value for a single signature in the aggregate signature with noticeable probability will directly result in an attack on unforgeability of the aggregate signature.

We first fix the exact distribution of signatures that must be aggregated, and then reason about the output of any given aggregation scheme on this input.
$\mathsf{GenSigs}(n, 1^\kappa)$:

1. For each $i \in [n]$, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and $r_i \leftarrow F_s$, and compute $R_i = r_i \cdot B$ and $\sigma_i = \mathsf{sk}_i \cdot \mathsf{RO}(\mathsf{pk}_i, R_i, 0) + r_i$

2. Output $(\mathsf{pk}_i, R_i, \sigma_i)_{i \in [n]}$

The $\mathsf{GenSigs}$ algorithm simply creates n uniformly sampled signatures on the message '0'.

Theorem 4.2.1. *Let* $(\mathsf{AggregateSig}, \mathsf{AggregateVerify})$ *characterize an aggregate signature scheme for* KeyGen, Sign, Verify *as per Schnorr with group* $(\mathbb{G}, B, q)$ *such that* $|q| = 2\kappa$. *Let* $\mathcal{Q}_V$ *be the list of queries made to* RO *by*

$$\mathsf{AggregateVerify}^{\mathsf{RO}}(\mathsf{AggregateSig}^{\mathsf{RO}}(\{\mathsf{pk}_i, R_i, \sigma_i\}_{i \in [n]}))$$

where $(\mathsf{pk}_i, R_i, \sigma_i)_{i \in [n]} \leftarrow \mathsf{GenSigs}(n, 1^\kappa)$. *Then for any* n, $\mathsf{max}((\Pr[(\mathsf{pk}_i, R_i, 0) \notin \mathcal{Q}_V])_{i \in [n]})$ *is negligible in* κ.

Proof. Let $\varepsilon = \max((\Pr[(\mathsf{pk}_i, R_i, 0) \notin \mathcal{Q}_V])_{i \in [n]})$, and let $j \in [n]$ be the corresponding index. We now define an alternative signature generation algorithm as follows,
$\mathsf{GenSigs}^*(n, j, \mathsf{pk}_j, 1^\kappa)$:

1. For each $i \in [n] \setminus j$, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and $r_i \leftarrow F_s$, and compute $R_i = r_i \cdot B$ and $\sigma_i = \mathsf{sk}_i \cdot \mathsf{RO}(\mathsf{pk}_i, R_i, 0) + r_i$
2. Sample $\sigma_j \leftarrow F_s$ and $e_j \leftarrow F_s$
3. Set $R_j = \sigma_i \cdot B - e_j \cdot \mathsf{pk}_j$
4. Output $(\mathsf{pk}_i, R_i, \sigma_i)_{i \in [n]}$

Observe the following two facts about $\mathsf{GenSigs}^*$: (1) it does not use sk_j, and (2) the distributions of $\mathsf{GenSigs}$ and $\mathsf{GenSigs}^*$ appear identical to any algorithm that does not query $(\mathsf{pk}_i, R_i, 0)$ to RO. The first fact directly makes $\mathsf{GenSigs}^*$ conducive to an adversary in the aggregated signature game: given challenge public key pk, simply invoke $\mathsf{GenSigs}^*$ with $\mathsf{pk}_j = \mathsf{pk}$ to produce $(\mathsf{pk}_i, R_i, \sigma_i)_{i \in [n]}$ and then feed these to $\mathsf{AggregateSig}$[1]. The advantage this simple adversary is given by the probability that the verifier does not notice that that $\mathsf{GenSigs}^*$ did not supply a valid signature under pk^* to $\mathsf{AggregateSig}$, and we can quantify this using the second fact as follows:

$$\begin{aligned}
&\Pr[\mathsf{AggregateVerify}^{\mathsf{RO}}(\mathsf{AggregateSig}^{\mathsf{RO}}(\mathsf{GenSigs}^*(n, j, \mathsf{pk}_j, 1^\kappa))) = 1] \\
&= \Pr[\mathsf{AggregateVerify}^{\mathsf{RO}}(\mathsf{AggregateSig}^{\mathsf{RO}}(\mathsf{GenSigs}(n, 1^\kappa))) = 1] - \Pr[(\mathsf{pk}_i, R_i, 0) \in \mathcal{Q}_V] \\
&= 1 - \Pr[(\mathsf{pk}_i, R_i, 0) \in \mathcal{Q}_V] \\
&= 1 - (1 - \varepsilon) = \varepsilon
\end{aligned}$$

Assuming unforgeability of the aggregated signature scheme, ε must be negligible. □

Corollary 4.2.2. *The output of* $\mathsf{AggregateSig}$ *must be at least* $2n\kappa - O(\log \kappa)$ *bits on average.*

This follows from the fact that the verifer must receive all n nonces $\{R_i\}_{i \in [n]}$, and they are sampled uniformly from the space $\mathbb{G}^n$.

4.3 Aggregating Schnorr Signatures With Tight Security

We now explore the application of Fischlin's transformation to our Sigma protocol in order to construct a non-interactive proof of knowledge that enjoys a tight reduction (yielding *provably secure* parameters, unlike Fiat-Shamir) while achieving a compression rate that can be arbitrarily close to 2. However the proximity to factor 2 compression comes at the expense of prover computation.

[1] If necessary, intercept $(\mathsf{pk}_j, R_j, 0)$ queried by $\mathsf{AggregateSig}$ to RO, and respond with e_j as set by $\mathsf{GenSigs}^*$

Concretely as per [CGKN21, Figure 2] aggregating EdDSA[2] signatures naively with Fischlin's transformation incurs an amortized cost of 4.2ms per signature when compressing by a factor of 1.33, and 39.7ms for factor 1.81 compression. This is multiple orders of magnitude slower than the Fiat-Shamir compiled proof (which incurs a fraction of a microsecond per signature on the same hardware) and processing even hundreds of signatures at once becomes prohibitively expensive.

Related Work. Recently, Chen and Zhao [CZ22] showed that the Fiat-Shamir compiled construction of Chalkias et al. can be proven secure with a tight reduction in the Random Oracle and Algebraic Group Model [FKL18]. While such a proof can build confidence in the Fiat-Shamir construction in that it rules out attacks by algebraic adversaries, the aim of this thesis is to be more conservative with assumptions, i.e. we consider security against any attack in the random oracle model. Interestingly, Chen and Zhao also showed that in the related (but incomparable) model of *sequential* aggregation [LMRS04] it is possible to prove a Fiat-Shamir compiled construction secure with a tight reduction in the random oracle model alone.

Faster Straight-Line Extraction. In this section we will develop the tools to substantially speed up the aggregation of EdDSA signatures with straight-line extraction in the random oracle model. Our improved aggregation algorithm is up to 200× faster for practically relevant parameters, and potentially within the performance envelope of real-world applications.

Applying Fischlin's Transformation. We can directly apply Fischlin's transformation to the Sigma protocol Σ_{aggr} to obtain a non-interactive proof. In particular, a 'base unit' of the proof is a challenge-response pair (e_j, z_j) such that $H(\mathsf{prefix}, e_j, z_j) = 0$ where H is an ℓ-bit random oracle, and this unit is repeated r times in order to achieve a κ-bit soundness level. These parameters are set so that a successful prover must query the random oracle with at least n accepting transcripts except with probability $2^{-\kappa}$.

Breaking down the cost. We can express the prover's computation cost in producing a proof as $\mathsf{T}_{\mathsf{Agg}} \cdot \mathsf{C}_{\mathsf{qry}}$, where $\mathsf{T}_{\mathsf{Agg}}$ is the prover query complexity, i.e. the number of (e, z) values the prover queries to the random oracle, and $\mathsf{C}_{\mathsf{qry}}$ is the cost of generating each (e, z) value. We discuss below how to improve on both of these dimensions.

[2]We use EdDSA to refer to Ed25519 [BDL+12] in particular, which is believed to instantiate a 128-bit security level.

Algorithm PolyEval

This algorithm is parameterized by a finite field $\mathbb{Z}_q$ where q is prime, a primitive k^{th} root of unity $\omega \in \mathbb{Z}_q$, and a degree n polynomial $f \in \mathbb{Z}_q[X]$. For simplicity we assume that k divides n. The output of this algorithm is a list of points $\{(x_i, f(x_i))\}_{i\in[k]}$.

PolyEval(q, k, f, n):

1. Parse the coefficients of f, with c_i as the coefficient of x^i
2. For each $i \in [0..k-1]$, define polynomial $f_i(x) = \sum_{j\in[0..n/k]} x^j \cdot c_{jk+i}$
3. Sample $\alpha \leftarrow \mathbb{Z}_q^*$ and for each $i \in [0..k-1]$ compute $\boldsymbol{\alpha}_i = f_i(\alpha^k)$
4. Define the degree $k-1$ polynomial $h(x) = \sum_{i\in[0..k-1]} \boldsymbol{\alpha}_i x^i$
5. Let points denote the (initially empty) list of output points
6. For each $i \in [0..k-1]$, append $(\alpha \cdot \omega^i, h(\alpha \cdot \omega^i))$ to points
7. Output points

Figure 4.2: Improved Polynomial Evaluation

4.3.1 Reducing $\mathsf{C}_{\mathsf{qry}}$ via Improved Polynomial Evaluation

The efficiency of polynomial evaluation algorithms is usually tied to the degree of the polynomial being evaluated. In our case, the degree of the polynomial corresponds to the number of signatures being aggregated. As the signature batch size can be small in practice (eg. number of transactions in a block, which is around 2000 for Bitcoin [Blo]) asymptotically efficient polynomial evaluation algorithms [vzGG13, BCKL21] may not be relevant to our setting.

Theorem 4.3.1. *Given a prime q, degree n polynomial $f \in \mathbb{Z}_q[X]$, and primitive k^{th} root of unity $\omega \in \mathbb{Z}_q$, Algorithm* PolyEval *outputs a list of k distinct points that lie on f at a cost of $k^2 + n + 2\log k$ multiplications and $k(k-1) + n$ additions in $\mathbb{Z}_q$.*

Proof. We begin by showing correctness. It suffices to show that for any $\alpha \in \mathbb{Z}_q^*$, the corresponding polynomial h agrees with f on the points $\{\alpha \cdot \omega^j\}_{j\in[0..k-1]}$. First we establish that $f(x) = \sum_{i\in[0..k-1]} x^i f_i(x^k)$ for every $x \in \mathbb{Z}_q$—this follows from the definition of f_i. Next we

use the fact that ω is a k^{th} root of unity to simplify the expansion of $f(\alpha \cdot \omega^j)$ as follows:

$$f(\alpha \cdot \omega^j) = \sum_{i \in [0..k-1]} (\alpha \cdot \omega^j)^i f_i((\alpha \cdot \omega^j)^k) = \sum_{i \in [0..k-1]} (\alpha \cdot \omega^j)^i f_i(\alpha^k)$$
$$= \sum_{i \in [0..k-1]} (\alpha \cdot \omega^j)^i \boldsymbol{\alpha}_i = h(\alpha \cdot \omega^j)$$

Now we count the number of multiplications in $\mathbb{Z}_q$ used by PolyEval. Step 3 requires computing α^k ($2 \log k$ multiplications by repeated squaring) and evaluating k degree n/k polynomials. Assuming we naively make use of Horner's rule (n/k multiplications and as many additions per polynomial), it costs n multiplications and n additions in $\mathbb{Z}_q$ to evaluate these polynomials, for a total of $n + 2 \log k$ $\mathbb{Z}_q$ multiplications and n additions induced by Step 3. Finally, in Step 6 we require k multiplications to generate each $\alpha \cdot \omega^i$, and we can evaluate the degree $k-1$ polynomial h at k points using Horner's rule, bringing the cost for this step to k^2 multiplications and $k(k-1)$ additions in $\mathbb{Z}_q$. Across all steps, the total number of operations required are $k^2 + n + 2 \log k$ multiplications, and $k(k-1) + n$ additions in $\mathbb{Z}_q$. This proves the theorem. □

While this is a significant improvement over the naive polynomial evaluation algorithm (which requires nk $\mathbb{Z}_q$ multiplications), in our application we need to evaluate f over a large set of points, and PolyEval only produces a batch of k evaluations. A simple extension to produce a batch of say $m \cdot k$ evaluations is to invoke PolyEval m times independently. However it is possible that there may be some redundancy across the multiple evaluations, i.e. independent instances may evaluate f at the same point. We show via Lemma 4.3.2 and Corollary 4.3.3 that for the parameters relevant to our setting, the probability of there being any redundancy is negligible.

Lemma 4.3.2. *The probability that m independent invocations of PolyEval with the same polynomial $f \in \mathbb{Z}_q[X]$ and parameter k will output fewer than $m \cdot k$ distinct points (i.e. repeat at least one point) is at most $m^2 k / 2q$*

Proof. In the event of a repetition, two independent invocations sample α and α' that induce at least one common point, i.e. $\alpha \cdot \omega^i = \alpha' \cdot \omega^j$ for some $i, j \in [k]$. Rearranging the terms, we see that it must be the case that the ratio α/α' is an integer power of ω. Note that there are exactly k integer powers of ω in $\mathbb{Z}_q$, i.e. the multiplicative subgroup that it generates. For any fixed $x \in \mathbb{Z}_q^*$, the probability that a uniformly chosen $y \in \mathbb{Z}_q$ is such that the ratio y/x lands in this subgroup is k/q.

If we denote α_i as the α value sampled by the i^{th} invocation of PolyEval and correspond-

ingly $\mathbf{A_i} = \{\alpha_i \cdot \omega^j\}_{j\in[0..k-1]}$, we can therefore bound the event of a repetition as follows:

$$\begin{aligned}\Pr[\exists i,j \in [m] : i \neq j, \mathbf{A_i} \cap \mathbf{A_j} \neq \varnothing] &= \Pr\left[\bigvee_{i,j\in[m]} \mathbf{A_i} \cap \mathbf{A_j} \neq \varnothing\right] \\ &\leq \sum_{i\in[m-1]} \sum_{j\in[i+1..m]} \Pr[\mathbf{A_i} \cap \mathbf{A_j} \neq \varnothing] \\ &\leq \sum_{i\in[m-1]} \sum_{j\in[i+1..m]} \frac{k}{q} \leq \frac{m^2 k}{2q}\end{aligned}$$

This proves the lemma. □

Corollary 4.3.3. *Given a parameter κ, if $q \in \Omega(2^\kappa)$ and $m, k \in \mathsf{poly}(\kappa)$, the probability that m independent invocations of* PolyEval *with the same polynomial will result in a redundant evaluation is negligible in κ.*

Efficiency. As per Theorem 4.3.1, PolyEval achieves the best improvement when $k \approx \sqrt{n}$. In this case, evaluating a degree n polynomial at $\sqrt{n}$ points costs roughly $2n$ multiplications, which is a factor $\sqrt{n}/2$ improvement over the naive method. This improvement is subject to the availability of appropriate k in the field in question. The setting that we consider in this thesis involves the EdDSA signature scheme, which uses Curve25519 [Ber06], which in turn is of order q such that $q-1$ is divisible by 4, 3, and 11. Given that we are interested in $n < 2^{12}$ or so, we are able to find a nearly optimal k for for any value of n in our range. We plot the improvement achieved by PolyEval in Figure 4.3.

Comparison with ECFFT. The very recent work of Ben-Sasson et al. [BCKL21] introduces a method to enable an FFT-like recursive evaluation of a polynomial in any arbitrary $\mathbb{Z}_q$, by using isogenies of elliptic curves. Their algorithm achieves impressive asymptotic as well as concrete performance in the preprocessing model, and can be applied to our setting. In particular, their $O(n \log^2(n))$ complexity is asymptotically superior to our $O(n^{1.5})$ PolyEval algorithm. However for our parameter range, we find our PolyEval algorithm to perform better, as we show in Figure 4.4.

Further Applications

The algorithm PolyEval is generally useful in settings where one has to evaluate a degree n polynomial in $\mathbb{Z}_q$, where n ranges from say 2^5 to 2^{14}, and $q-1$ is 'slightly smooth', i.e. there are enough $k \approx \sqrt{n}$ values that divide $q-1$. Such settings include the base fields of common elliptic curves such as Curve25519 (discussed in this paper in the context of EdDSA), and

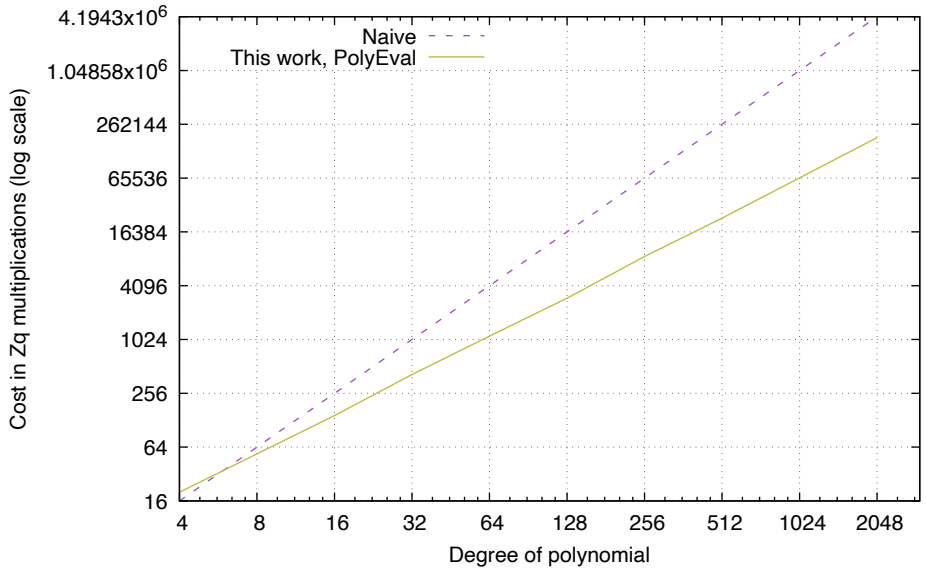

Figure 4.3: This graph plots the computation cost of evaluating a polynomial of degree n up to 2^{12} at n points in $\mathbb{Z}_q$, where q is the order of the elliptic curve Curve25519 used for EdDSA. The cost is derived analytically.

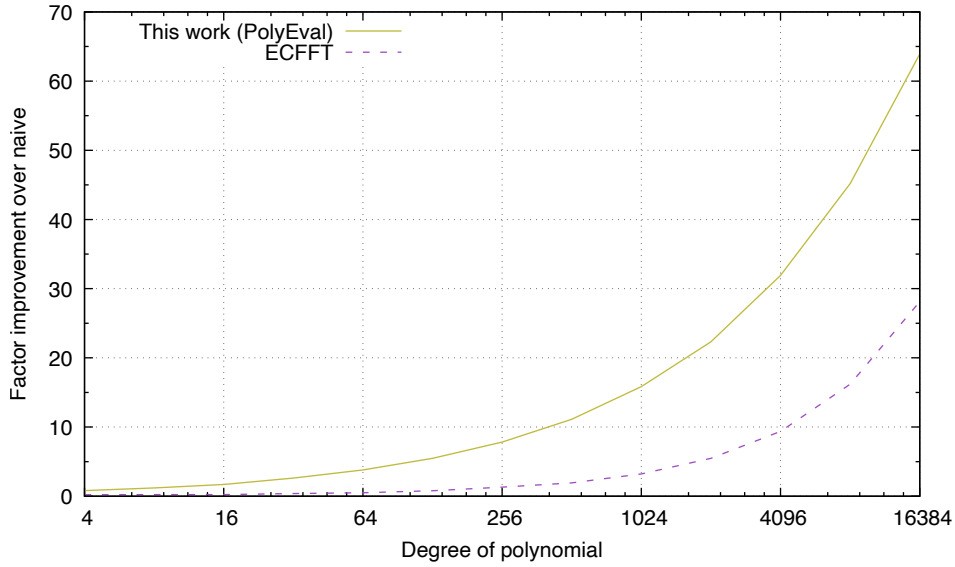

Figure 4.4: This graph plots the factor improvement over the naive method, in evaluating a polynomial of degree n up to 2^{14} at n points in $\mathbb{Z}_q$, where q is the order of the BN-254 elliptic curve. The improvement factor for ECFFT is taken from a public Rust implementation [wbo]. We did not re-implement PolyEval for this curve, however our Rust implementation for Ed25519 is faithful to our analytical estimate, and so we derived the improvement factor for PolyEval analytically.

secp256k1 (used by Bitcoin and others for ECDSA). We describe some of these settings where PolyEval can be relevant in this section.

Threshold Cryptography. A common method to protect signing/encryption keys is to distribute them across a number m of devices, so that reconstructing or operating with the key requires a threshold t of the devices to cooperate. This is typically done by using Shamir's secret sharing in the base field of the elliptic curve, i.e. defining a degree $t-1$ polynomial f such that $f(0) = \mathsf{sk}$ encodes the secret key, and each party P_i receives $f(i)$. When t is in the range of 2^5 to 2^{14}, PolyEval can speed up the generation of these shares for threshold versions of EdDSA and ECDSA keys.

Verifiable Secret Sharing and Beyond. There are numerous constructions to upgrade the security of secret sharing schemes to tolerate a malicious dealer and participants, i.e. *verifiable* secret sharing (VSS). Simple VSS schemes such as Feldman's [Fel87a] for groups where the discrete logarithm assumption is assumed to hold form the basis for distributed key generation protocols [Ped91] for ECDSA/EdDSA. VSS can also form the basis for *verifiable encryption* [CD00], where a ciphertext can be verified to encrypt the discrete logarithm of a public point (say encrypt the secret component of an EdDSA/ECDSA public key), when it is combined with MPC-in-the-head techniques [TZ21]. In this case, the degree of the polynomial corresponds to the number of 'transcripts' that must be checked, which for a 128 or 256 bit security level falls within the previously mentioned range for which PolyEval provides significant savings.

4.3.2 Improving Prover Query Complexity $\mathsf{T}_{\mathsf{Agg}}$

First we note that we can parameterize the proof of work via a better analysis tailored to the signature aggregatio setting, which yields an improvement of 2 to 8× in the hardness setting for the proof-of-work problem. Intuitively this is because the direct application of Fischlin's transform results in repeating a base unit sufficiently many times for the desired soundness level, whereas one can prove better parameters by directly analyzing the final construction, i.e. the event that a malicious prover finds r inversions within n queries.

Our idea. We change the underlying 'proof of work problem' solved by the prover from finding r inversions to finding an r-collision. In particular the prover now searches for $(e_j, z_j)_{j\in[r]}$ such that $H(\mathsf{prefix}, e_1, z_1) = \cdots = H(\mathsf{prefix}, e_r, z_r)$, where H is a random oracle with output bit length $\ell \geq (\kappa + r\log_2(n) - \log(r!))/(r-1)$. This yields a ≈ 1.5 to 2× improvement in $\mathsf{T}_{\mathsf{Agg}}$ corresponding to the ratio of the costs of finding an r-collision to that

of finding r inversions at the same security level (even with the improved analysis).

We give the full protocol and justify its parameterization below. We give the concrete query complexity improvements in Table 4.3.4, although we defer a more precise analytical justification of why finding an r-collision is faster than finding an equivalent number of inversions at the same security level to Section 4.4.3.

n	r	Collision (This work)	Inversion	Improvement
1024	8	9.33×10^7	2.68×10^8	2.8
512	8	5.11×10^7	1.34×10^8	2.6
512	16	2.95×10^5	5.24×10^5	1.7
1024	32	3.55×10^4	6.55×10^4	1.8
256	16	1.57×10^5	2.62×10^5	1.6
512	32	1.86×10^4	3.28×10^4	1.7
128	16	8.36×10^4	1.31×10^5	1.5
256	32	9.80×10^3	1.64×10^4	1.6
32	8	2.53×10^6	8.39×10^6	3.3
64	16	2.38×10^4	6.55×10^4	2.7
128	32	5.19×10^3	8.19×10^3	1.5

Table 4.3.4: Prover/aggregator query complexity $\mathsf{T}_{\mathsf{Agg}}$ when using a collision based predicate to aggregate n signatures, as opposed to inversions (with a tighter parameterization than [CGKN21]), for a range of r parameters. Expected running times are derived analytically [vM39, Pre93]

Caveat: Memory Complexity. We note that keeping track of collisions consumes more memory—$O(\mathsf{T}_{\mathsf{Agg}})$—than the inversion construction which only needs $O(\kappa)$. In practice, however, this is quite a small amount (up to 30MB for benchmarked parameters), as shown in Table 4.3.5 below.

Further Applications. The superior combinatorial characteristics of the collision problem over the inversion problem has interesting implications for the computation complexity of straight-line extraction even in the zero-knowledge setting. In Sections 4.4.1 and 4.4.3, we show how to improve the prover's query complexity when compiling *any* standard Sigma

n	r	Expected Memory Usage
1024	8	8.9GB
512	8	4.9GB
512	16	28.3MB
1024	32	3.4MB
256	16	15MB
512	32	1.7MB
128	16	8MB
256	32	0.9MB
32	8	242MB
64	16	2.2MB
128	32	0.5MB

Table 4.3.5: Prover/aggregator memory complexity when using a collision based predicate to aggregate n signatures for a range of r parameters, for a naive implementation. Derived analytically [vM39, Pre93]

protocol to a NIZKPoK by $10-15\%$, and for some special Sigma protocols by up to a factor of 2. The latter is particularly significant as it matches a new lower bound that we prove.

4.3.3 Putting It Together – Improved EdDSA Aggregation

We combine our improvements to $\mathsf{T}_{\mathsf{Agg}}$ and $\mathsf{C}_{\mathsf{qry}}$ to obtain an EdDSA signature aggregation algorithm π_{Aggr} with substantially improved prover computation complexity, which we give below in Figure 4.5. We further justify its performance improvements with our benchmarks in Table 3.2.3.

Theorem 4.3.6. *Protocol π_{Aggr} is a proof of knowledge for the relation R_{Agg} with straight-line extraction in the random oracle model.*

Proof. We know from Theorem 4.1.1 that the underlying Sigma protocol is n-special sound, which implies that once a malicious prover has queried n accepting transcripts to the random oracle, the entire witness can be extracted. It therefore suffices to analyze the smallest ℓ that guarantees that a cheating prover is unable find an r-collision within $\leq n$ queries except with

probability $2^{-\kappa}$. The number of events (i.e. assignments of random oracle outputs) in which the first n queries to an ℓ-bit random oracle contain an r-collision is at most:

$$\binom{n}{r} \cdot 2^{\ell} \cdot (2^{\ell})^{(n-r)}$$

Here $\binom{n}{r}$ counts the number of combinations of indices to 'plant' an r-collision, there are 2^{ℓ} values that the collision can take, and there are $(2^{\ell})^{(n-r)}$ assignments of the remaining $n - r$ indices. This term is not tight since we double-count $r + 1$ collisions, triple count $r + 2$ collisions, etc. but their impact is minimal. Since there are a total of $2^{n\ell}$ equally likely possible output assignments to n random oracle queries, we have that:

$$\Pr[r\text{-collision within the first } n \text{ steps}] \leq \frac{\binom{n}{r} \cdot 2^{\ell} \cdot (2^{\ell})^{(n-r)}}{2^{n\ell}}$$

It remains to examine the constraint on ℓ that will induce the above probability to be upper bounded by $2^{-\kappa}$:

$$\begin{aligned}
\frac{\binom{n}{r} \cdot 2^{\ell} \cdot (2^{\ell})^{(n-r)}}{2^{n\ell}} &\leq 2^{-\kappa} \\
\binom{n}{r} 2^{\ell(1+n-r-n)} &\leq 2^{-\kappa} \\
\frac{n^r}{r!} 2^{\ell(1-r)} &\leq 2^{-\kappa} \\
2^{\ell(1-r)} &\leq r! \cdot 2^{-(\kappa + r\log_2(n))} \\
&\leq 2^{-(\kappa + r\log_2(n) - \log_2(r!))} \\
2^{\ell(r-1)} &\geq 2^{\kappa + r\log_2(n) - \log_2(r!)} \\
\ell &\geq (\kappa + r\log_2(n) - \log_2(r!))/(r-1)
\end{aligned}$$

which is precisely the constraint adhered to by ℓ in π_{Aggr}. □

4.4 Applying the Collision Predicate to NIZKPoK

We apply the principle of replacing hash inversions in Fischlin's transformation with hash collisions to the original NIZKPoK transform, and observe improved prover query complexity in this setting as well. We begin by considering the hash collision predicate as a *drop-in replacement* to any Sigma protocol for which Fischlin's transformation can be applied, and observe an $11 - 15\%$ improvement in the prover's query complexity.

To our knowledge this is the best query complexity achieved for NIZKs so far, however a natural question is to ask to what extent such techniques can be extended. To this end,

Protocol π_{Aggr}

The prover P and verifier V are both given the public instance $(\mathsf{pk}_i, m_i, R_i)_{i\in[n]} \in (\mathbb{G} \times \{0,1\}^* \times \mathbb{G})^n$ while the prover also has witness $(s_i)_{i\in[n]} \in \mathbb{Z}_q^n$ for the statement $s_i \cdot G = H_{\mathsf{Sch}}(\mathsf{pk}_i, R_i, m_i) \cdot \mathsf{pk}_i + R_i \ \forall i \in [n]$. Both parties have access to an ℓ-bit Random Oracle $H : \{0,1\}^* \mapsto \{0,1\}^\ell$ where $\ell \geq (\kappa + r\log_2(n) - \log_2(r!))/(r-1)$.

$\mathsf{P}^H((\mathsf{pk}_i, m_i, R_i, s_i)_{i\in[n]})$:

1. Find k closest to $\sqrt{n}$ such that $k \mid q-1$
2. Set $\boldsymbol{a} = (\mathsf{pk}_i, m_i, R_i)_{i\in[n]}$, and define polynomial $f(x) = \sum_{i\in[n]} x^i \cdot s_i$
3. Initialize $\mathcal{Z} = \emptyset$ and do the following until an output is produced:
   (a) Obtain $\mathsf{points} \leftarrow \mathsf{PolyEval}(q, k, f, n)$ and append each $(e, z) \in \mathsf{points}$ to $\mathcal{Z}$
   (b) If $\exists (e_1, z_1), (e_2, z_2), \cdots, (e_r, z_r) \in \mathcal{Z}$ such that
   $$H(\boldsymbol{a}, e_1, z_1) = H(\boldsymbol{a}, e_2, z_2) = \cdots = H(\boldsymbol{a}, e_r, z_r)$$
   then set $\boldsymbol{e} = (e_i)_{i\in[r]}$ and $(z_i)_{i\in[r]}$ and output $\pi = (\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z})$

$\mathsf{V}^H((\mathsf{pk}_i, m_i, R_i)_{i\in[n]}, \pi)$:

1. Parse $(\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z}) = \pi$, and $(e_i)_{i\in[r]} = \boldsymbol{e}$, and $(z_i)_{i\in[r]} = \boldsymbol{z}$.
2. Check that $H(\boldsymbol{a}, e_1, z_1) = H(\boldsymbol{a}, e_2, z_2) = \cdots = H(\boldsymbol{a}, e_r, z_r)$
3. For each $i \in [n]$, compute $S_i = H_{\mathsf{Sch}}(\mathsf{pk}, R, m) \cdot \mathsf{pk} + R$
4. For each $i \in [r]$, check that $z_i \cdot G = \sum_{i\in[n]} e^i \cdot S_i$, aborting with output 0 if not
5. Accept by outputting 1

Figure 4.5: Collision Based Aggregation of n Signatures

we show a lower bound on the query complexity of *any* NIZK that has a straight-line non-programming extractor in Section 4.4.2. We find that Fischlin's construction (which is the most query efficient straight-line extractable scheme) never meets this lower bound for any non-trivial parameters.

We show in Section 4.4.3 that it is indeed feasible to meet this lower bound for some non-trivial parameters, by means of a new transformation based on our collision predicate. Unfortunately this transformation only applies to a special class of Sigma protocols that have an r-simulatability property. We show in Appendix A.2 how to construct such a Sigma protocol by extending Schnorr's proof of knowledge of discrete logarithm.

4.4.1 Unconditionally Improving Fischlin's Query Complexity

Recall that the prover in Fischlin's transformation is required to invert a fixed target of the random oracle. In particular, a proof consists of a base unit where the prover is required to find a Sigma protocol transcript (a, e, z) such that $H(\mathsf{prefix}, a, e, z) = 0^\ell$, and this unit is repeated r times to achieve $\kappa = r \cdot \ell$ bits of security. We can replace this inversion based unit by a collision based one as follows: the prover is required to find a pair of independent transcripts (a_1, e_1, z_1) and (a_2, e_2, z_2) such that $H(\mathsf{prefix}, a_1, e_1, z_1) = H(\mathsf{prefix}, a_2, e_2, z_2)$. Note that just as in the case of Fischlin, prefix includes a_1, a_2 to prevent trivial attacks. Additionally, the output length of the hash function is 2ℓ, i.e. doubled as compared to the inversion predicate.

Security. Upon fixing prefix, a prover is successful in finding an accepting pair (a_1, e_1, z_1) and (a_2, e_2, z_2) in their first attempt with probability no more than $2^{-2\ell}$. Repeating this base unit $r/2$ times achieves security $2\ell \cdot r/2 = \kappa$ bits.

Efficiency. A base unit of the collision based construction is equivalent to two base units of the inversion construction; in both cases two Sigma protocol transcripts are transmitted, and they achieve 2ℓ bits of security. With regards to computation cost, both constructions have the same cost per query made to the random oracle (i.e. computing a fresh Sigma protocol response), and therefore the difference comes down to the number of queries made per proof, i.e. the prover query complexity.

What query complexity does this induce? Consider $\mathcal{Z}_1, \mathcal{Z}_2$ to be domains from which (e_1, z_1) and (e_2, z_2) are drawn respectively, and observe that $\mathcal{Z}_1, \mathcal{Z}_2$ are entirely disjoint when $a_1 \neq a_2$. If we consider $(\mathsf{prefix}, a_1, e_1, z_1)$ and $(\mathsf{prefix}, a_2, e_2, z_2)$ to be the 'left' and 'right' halves of the collision respectively, this means that any given $(\mathsf{prefix}, a_i, e_i, z_i)$ can be a candidate pre-image for either the left or right half, but not both. This is because any given e_i, z_i can

be a verifying transcript with at most one of a_1 or a_2. This task therefore becomes that of finding a *chosen prefix* collision [SLdW07]. The combinatorics of chosen prefix collisions are considerably more complex to analyze than regular collisions, making the derivation of the exact query complexity of the above construction difficult. We instead measure the query complexity induced by this predicate empirically, and report on the results in Table 4.4.1.

As our experiments show, this chosen prefix collision predicate works for the exact same Sigma protocols as Fischlin's transformation, and improves on its query complexity. A natural question for future work is if we can obtain further improvements by considering multicollisions rather than pairs of collisions.

		Fischlin	Pairwise collisions		
r	ℓ	Expected queries	ℓ	Exp queries	Improvement
8	2^{16}	64,877	2^{32}	58,190	1.11
10	2^{13}	8,233	2^{26}	7,293	1.13
12	2^{11}	2,038	2^{22}	1,824	1.12
14	2^{9}	509	2^{18}	448	1.13
16	2^{8}	267	2^{16}	232	1.15

Table 4.4.1: Comparing the computation cost of Fischlin's approach to our chosen prefix, pairwise collision approach. The reported value is the expected number of queries for finding either one preimage, or 2 collisions taken over 500-2000 experiments. Parameters for r and ℓ are set for the same 128 bit security.

4.4.2 Lower Bound on Prover Query Complexity

Fischlin [Fis05] proved via a meta reduction that any NIZKPoK scheme (with a non-programming extractor) for a language with a hard instance generator, must have a super-logarithmic number of queries V in κ made by the verifier to the random oracle. Fischlin's proof demonstrated asymptotic bounds due to its reliance on the hardness of the underlying language; in this work we are concerned with tight parameters for concrete security as guaranteed in the random oracle model, independently of the hardness of the underlying language. We therefore initiate a study of concrete query complexity, in particular we express this as the optimal prover query complexity P upon fixing V.

Caveat. We make a simplifying assumption, namely that the language L has a hard

instance generator $\mathcal{I}$ such that the probability that any PPT algorithm is able to find a witness w for theorem $x \leftarrow \mathcal{I}(\kappa)$ is bounded by $\varepsilon_\kappa \ll 2^{-\kappa}$.

This assumption frequently does not hold as in practice one can instantiate the NIZKPoK with a concrete soundness level comparable to the hardness of instances generated by $\mathcal{I}$, however making this simplification allows us to focus on the random oracle query complexity of the NIZKPoK (which is given by parameters independent of the language) without having to account for concrete hardness of the language (which is very specific to each language and seldom leveraged by the extractor of a NIZKPoK scheme).

We begin with the following lemma, which is a tightening of [Fis05, Proposition 2]:

Lemma 4.4.2. *If* (P, V) *is a straight-line extractable NIZKPoK scheme for language* L *in the random oracle model with the following characteristics for security parameter* κ*:*

- *Perfect zero-knowledge simulator* $\mathcal{S}$
- ℓ*-bit output random oracle* H
- P *queries made by* P *to* H *in generating a proof*
- *Probability* $p_C > 0$ *of producing an accepting proof*
- V *queries made by deterministic* V *to* H *in verifying a proof, is a strict subset of the queries made by* P
- *Non-programming extractor* Ext *with error* $\leq 2^{-\kappa}$ *for an adversary that makes* $\leq V$ *queries to the random oracle*

Then it must hold that:

$$\binom{P}{V} \geq \frac{p_C}{2^{-\kappa} + \varepsilon_\kappa}$$

Proof. The idea is to show that if $\binom{P}{V}$ is too small, then a malicious prover can succeed in producing a verifying proof by just guess the queries that V would make in verifying a proof, and simulating the remaining ones. This means an extractor should be able to produce a witness using just the queries that V makes (since those are the only queries that this malicious prover P makes) and this contradicts the hardness of the language.

We begin by constructing a new Prover algorithm P' which internally runs P, but simulates most of the random oracle calls for P and only makes a total of V external calls to the real oracle H: $\mathsf{P}'^{H}(x, w, \mathsf{P})$:

1. Sample a set of *indices* $Q \subset [1, \ldots, P]$ such that $|Q| = V$
2. Define oracle $H'(v)$ as follows:
   - If this is the i^{th} invocation of the oracle and if $i \in Q$ then return $H(v)$
   - Otherwise return a uniform $\{0,1\}^\ell$
3. Obtain $\pi \leftarrow \mathsf{P}^{H'}(x, w)$ and output π

Let $\mathcal{Q}_P$ represent the queries to H' made by P. Assuming no redundant queries in $\mathcal{Q}_P$, we note that H' agrees with H on V randomly chosen queries, and the two are completely independent on all other inputs.

By completeness of (P, V), it holds with probability p_C that $\mathsf{V}^{H'}(x, \pi) = 1$. Our goal is to instead analyze the probability that $\mathsf{V}^{H}(x, \pi)$ accepts, i.e., the verifier who makes queries to the real external oracle H accepts π. Denote the queries made by V to H' as $\mathcal{Q}_V$. Given that $\mathcal{Q}_V \subset \mathcal{Q}_P$, and that H' agrees with H on V values,

$$\begin{aligned}\Pr\left[\mathsf{V}^{H}(x,\pi) = 1 : \pi \leftarrow \mathsf{P}'^{H}(x, w, \mathsf{P})\right] &= \Pr\left[\mathsf{V}^{H}(x,\pi) = 1 : \pi \leftarrow \mathsf{P}^{H}(x, w)\right] \\ &\quad \cdot \Pr[H'(x) = H(x), \forall x \in \mathcal{Q}_V] \\ &\geq p_C \cdot \binom{P}{V}^{-1}\end{aligned}$$

Recall that the extractor's error (in this case $2^{-\kappa}$) represents the difference between the probability that a malicious prover P^* is able to produce a proof π, and the probability that the extractor Ext is able to produce a witness w for x when given the proof π and list of queries made by P^* in its production. Note that P' only queries $\mathcal{Q}_V$ to H, and so the set $\mathcal{Q}_V$ fully characterizes the list of queries made by the malicious prover. We therefore determine that:

$$\Pr[w \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q}_V) : \pi \leftarrow \mathsf{P}'^{H}(x, w, \mathsf{P})]$$

$$\geq \Pr\left[\mathsf{V}^{H'}(x,\pi) = 1 : \pi \leftarrow \mathsf{P}'^{H}(x, w, \mathsf{P})\right] - 2^{-\kappa}$$

$$\geq p_C \cdot \binom{P}{V}^{-1} - 2^{-\kappa}$$

As a final step, we replace $\pi \leftarrow \mathsf{P}'^{H}(x, w, \mathsf{P})$ by $(\pi, H) \leftarrow \mathcal{S}(x)$ to remove reliance on the witness w. Note that these two distributions of (π, H) are identical due to the fact that when P' outputs a proof, it is identically distributed to the output of honest P, and that the perfect

simulation is distributed identically to the output of honest P. The set $\mathcal{Q}_V$ is fully specified by x, π, H as we show below.
$\mathcal{A}(x)$:

1. Compute $(\pi, H) \leftarrow \mathcal{S}(x)$
2. Construct $\mathcal{Q}_V$ by collecting the queries to H made by $\mathsf{V}^H(x, \pi)$
3. Output $\pi, \mathcal{Q}_V$

Firstly due to the perfect simulation we note that

$$\begin{aligned}\Pr[w \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q}_V) : (\pi, \mathcal{Q}_V) \leftarrow \mathcal{A}(x)] &= \Pr[w \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q}_V) : \pi \leftarrow \mathsf{P}'^H(x, w, \mathsf{P})] \\ &\geq p_C \cdot \binom{P}{V}^{-1} - 2^{-\kappa}\end{aligned}$$

Second we note that $w \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q}_V) : (\pi, \mathcal{Q}_V) \leftarrow \mathcal{A}(x)$ constitutes a PPT adversary that finds a witness for any $x \in L$. Since L has a hard instance generator $\mathcal{I}$ that admits a maximum advantage of ε_κ, for $x \leftarrow \mathcal{I}(\kappa)$ it holds that

$$\varepsilon_\kappa \geq \Pr[w \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q}_V) : (\pi, \mathcal{Q}_V) \leftarrow \mathcal{A}(x)] \geq p_C \cdot \binom{P}{V}^{-1} - 2^{-\kappa}$$

Rearranging, we have that

$$\binom{P}{V} \geq \frac{p_C}{2^{-\kappa} + \varepsilon_\kappa}$$

and this proves the lemma. □

We can use the above lemma to derive the optimal prover query complexity for proofs that are non-trivially secure, i.e. when $V \ll \binom{P}{V}$. We define $P_{\mathsf{OPT}}(V)$ to be the smallest prover query complexity for a given verifier query complexity V.

Corollary 4.4.3. *If (P, V) is a perfectly complete straight-line extractable NIZKPoK scheme for a ε_κ-hard language L in the random oracle model with all the characteristics required by Lemma 4.4.2 with the additional constraint that $V < \kappa$ and $2^{-\kappa} \gg \varepsilon_\kappa$, then the optimal prover query complexity is given by:*

$$P_{\mathsf{OPT}}(V) \approx (V! \cdot 2^\kappa)^{\frac{1}{V}}$$

Proof. As $2^{-\kappa} \gg \varepsilon_\kappa$, we make the approximation $2^{-\kappa} + \varepsilon_\kappa \approx 2^{-\kappa}$. From Lemma 4.4.2 we have that P_{OPT} is the smallest P such that $\binom{P}{V} \geq 2^\kappa$ since $p_C = 1$. Simplifying, we have that:

$$\begin{aligned} 2^\kappa &\leq \binom{P}{V} \\ 2^\kappa \cdot V! &= \prod_{i \in [0,V)} (P_{\mathsf{OPT}} - i) \\ &\approx (P_{\mathsf{OPT}})^V \end{aligned}$$

Upon rearranging the terms, we get the statement of the corollary. □

In subsequent text we drop the argument $[\kappa, V]$ when it is obvious. Note that P_{OPT} only characterizes the optimal prover query complexity for *perfectly complete* schemes. Since Lemma 4.4.2 accounts for schemes with arbitrary completeness errors, it is possible to amend Corollary 4.4.3 accordingly if desired. However we will see that P_{OPT} serves as a useful benchmark for our study. Interestingly Fischlin's scheme, which has the lowest prover query complexity in the literature, performs worse than P_{OPT} for all $V > 1$.

Claim 4.4.4. *Let r parameterize the number of repetitions of a Sigma protocol used to instantiate Fischlin's NIZK [Fis05] at a κ-bit security level. Then the average prover query complexity of the resulting scheme $\mathsf{T}_{\mathsf{Fis}}$ is a factor of $r/(r!)^{1/r}$ worse than the corresponding P_{OPT}. Therefore $\mathsf{T}_{\mathsf{Fis}} > P_{\mathsf{OPT}}$ for every $r > 1$.*

Proof. The average prover query complexity $\mathsf{T}_{\mathsf{Fis}}$ is given by the complexity of finding r inversions of the all-zero string of r independent κ/r-bit random oracles. This task requires $r \cdot 2^{\kappa/r}$ tries in expectation. Since $V = r$, the optimal prover complexity is given by $P_{\mathsf{OPT}} = (r! \cdot 2^\kappa)^{1/r}$. The ratio of the average prover complexity to the optimal is therefore:

$$\frac{\mathsf{T}_{\mathsf{Fis}}}{P_{\mathsf{OPT}}} = \frac{r \cdot 2^{\kappa/r}}{(r! \cdot 2^\kappa)^{1/r}} = \frac{r}{(r!)^{1/r}}$$

□

The ratio $\mathsf{T}_{\mathsf{Fis}}/P_{\mathsf{OPT}} = 1$ only when $r = 1$, which is of no use as the average complexity of computing a proof honestly matches the average complexity of forging a proof when $r = 1$. This ratio is $\sqrt{2} \approx 1.41$ when $r = 2$, and continues to increase as r grows, ultimately converging[3] at $e \approx 2.71$. Given this it is natural to ask, is it possible to meet P_{OPT} for any non-trivial parameters?

[3] $\lim_{r \to \infty} r/(r!)^{1/r} = e$

4.4.3 Special Case: $r+1$-Special Sound Sigma Protocols

Given a Sigma protocol that is $r+1$-special sound and r simulatable (i.e. given r challenges, a simulator can produce r accepting transcripts) we are able to apply a multicollision predicate and reduce the prover's query complexity as compared with Fischlin's inversion predicate even further—to the point where we can meet P_{OPT} for a non-trivial parameter range.

Note that we present a randomized construction here—this aspect is orthogonal to query complexity. The purpose is to avoid dependence on 'quasi-unique responses', which we will discuss in detail in Section 4.5.

We begin by refining the standard definition of Sigma protocols [Dam02] to incorporate a weaker notion of soundness and simulatability. This notion essentially requires (1) $r+1$-special soundness, which guarantees the success of an extractor upon being given $r+1$ accepting conversations that begin with the same first message, and (2) r-simulatability, which requires that for any statement, r accepting conversations (with the same first message) can be simulated for any r given challenges. We defer a formal definition to Appendix A.1, and give an instantiation based on Schnorr's PoK of discrete logarithm in Appendix A.2. We describe our NIZK transformation in Figure 4.6.

Theorem 4.4.5. *If Σ is a strongly $r+1$-special sound Sigma protocol and $\ell(r-1)=\kappa$, the protocol π_{NIZK} is a straight-line extractable NIZKPoK in the random oracle model, with an extractor that does not program the random oracle and achieves extraction error $Q/2^{\kappa}$ for an adversary making Q queries to the random oracle.*

Proof. (Sketch) We defer the full proof to Appendix A.1. Completeness follows from the pigeonhole principle, as any function that maps a domain of size $r \cdot 2^{\ell}$ to a range of size 2^{ℓ} will produce at least one r-collision. Zero-knowledge comes from the fact that the challenges $\boldsymbol{e}$ are distributed uniformly in $\{0,1\}^{t \cdot r}$, and the rest of the transcripts $\boldsymbol{a}, \boldsymbol{z}$ can be simulated by invoking $\mathcal{S}_{\Sigma}(x, r, \boldsymbol{e})$. Proof-of-knowledge follows from the fact that in order for an adversary to compute a proof by querying fewer than $r+1$ accepting Sigma protocol transcripts to H, the first r accepting transcripts it queries to H must all evaluate to the same ℓ-bit string. This happens with probability $(2^{-\ell})^{r-1} = 2^{-\kappa}$. □

Query Complexity. We make use of the analysis of multicollision running times by von Mises [vM39] and revisited by Preneel [Pre93, Appendix B].

Corollary 4.4.6. *[vM39][Pre93, Theorem B.2 and pg. 283] If T balls are randomly distributed over n urns, the number T required to have at least one urn with r balls with probability $1-\exp(-\alpha_r)$ is given by the following equation:*

$$T \cdot \exp\left(-\frac{T}{r \cdot n}\right) = \left(\alpha_r \cdot n^{(r-1)} \cdot r!\right)^{1/r}$$

Protocol π_{NIZK}

The prover P and verifier V are both given the statement x while the prover also has a witness w for the statement $x \in L$. Both parties have access to an ℓ-bit Random Oracle $H : \{0,1\}^* \mapsto \{0,1\}^\ell$. The underlying Strongly $r+1$-special sound sigma protocol is given by $\Sigma = ((P_{\Sigma,a}, P_{\Sigma,z}), \mathsf{V}_\Sigma)$. Define $t = \ell + \lceil \log r \rceil$.

$\mathsf{P}^H(x, w)$:

1. Run $P_{\Sigma,a}(x, w)$ to obtain $\boldsymbol{a}$ and state
2. Set $\mathcal{E} = \mathcal{Z} = \emptyset$ and do the following until an output is produced:
   (a) Uniformly sample $e \leftarrow \{0,1\}^t \setminus \mathcal{E}$
   (b) Set $z = P_{\Sigma,z}(\mathsf{state}, e)$ and append (e, z) to $\mathcal{Z}$ and e to $\mathcal{E}$
   (c) If $\exists (e_1, z_1), (e_2, z_2), \cdots, (e_r, z_r) \in \mathcal{Z}$ such that
   $$H(\boldsymbol{a}, e_1, z_1) = H(\boldsymbol{a}, e_2, z_2) = \cdots = H(\boldsymbol{a}, e_r, z_r)$$
   then set $\boldsymbol{e} = (e_i)_{i \in [r]}$ and $(z_i)_{i \in [r]}$ and output $\pi = (\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z})$

$\mathsf{V}^H(x, \pi)$:

1. Parse $(\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z}) = \pi$, and $(e_i)_{i \in [r]} = \boldsymbol{e}$, and $(z_i)_{i \in [r]} = \boldsymbol{z}$.
2. Check that $H(\boldsymbol{a}, e_1, z_1) = H(\boldsymbol{a}, e_2, z_2) = \cdots = H(\boldsymbol{a}, e_r, z_r)$
3. For each $i \in [r]$, check that $\mathsf{V}_\Sigma(x, (\boldsymbol{a}, e_i, z_i)) = 1$, aborting with output 0 if not
4. Accept by outputting 1

Figure 4.6: Collision Based NIZK

In order to obtain the time $\mathsf{T}_{\mathsf{Col}}$ required to find an r-collision in expectation, one must solve for T when the parameter $\alpha_r = 1$. Substituting $n = 2^{\kappa/(r-1)}$ for our context, we get that:

$$\mathsf{T}_{\mathsf{Col}} \cdot \exp\left(-\frac{\mathsf{T}_{\mathsf{Col}}}{r \cdot 2^{\kappa/(r-1)}}\right) = (2^\kappa \cdot r!)^{1/r} = P_{\mathsf{OPT}}$$

This equation is non-trivial to analyze relative to that of Fischlin, and so for ease of understanding we plot the ratio T/P_{OPT} for both π_{NIZK} and Fischlin's construction in Figure 4.7. This plot shows that for some reasonable parameterizations around $r \sim 5$, our construction achieves roughly 2x factor improvement in Prover complexity.

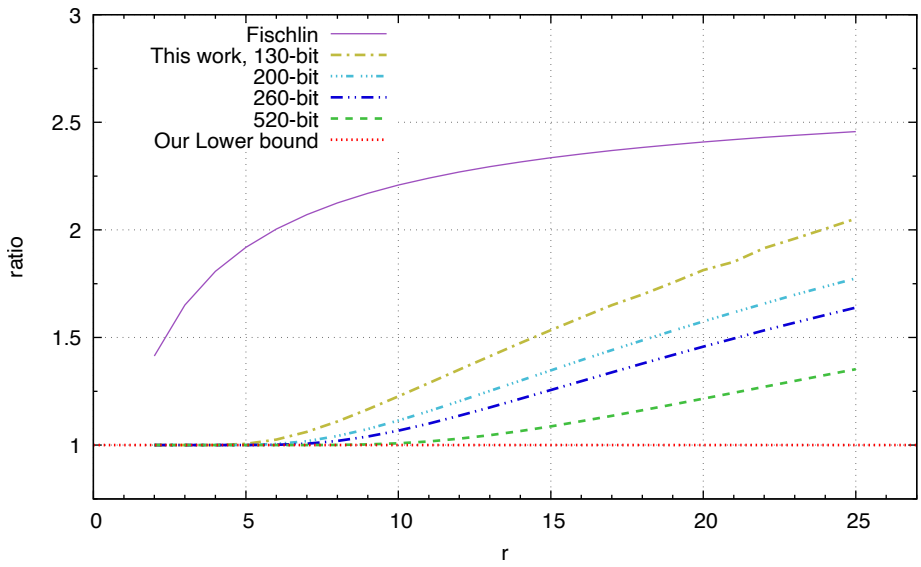

Figure 4.7: Ratio of prover query complexities $\mathsf{T}_{\mathsf{Col}}$ and $\mathsf{T}_{\mathsf{Fis}}$ to the optimal P_{OPT} (y-axis) for different r parameters (x-axis), where $\mathsf{T}_{\mathsf{Col}}[r]$ and $\mathsf{T}_{\mathsf{Fis}}[r]$ are the number of oracle queries required to compute a proof in expectation upon fixing parameter r. Note that $\mathsf{T}_{\mathsf{Col}}/P_{\mathsf{OPT}}$ depends on the security parameter, whereas $\mathsf{T}_{\mathsf{Fis}}/P_{\mathsf{OPT}}$ is essentially invariant of it. Consequently we plot $\mathsf{T}_{\mathsf{Col}}/P_{\mathsf{OPT}}$ for a range of security parameters, where "κ-bit Col" denotes a κ-bit security level.

Finally, we note that Figure 4.7 only plots the ratio of Fischlin/Collision/optimal but does not convey the actual prover query complexities at those parameter choices. Table 4.4.7 below shows the Prover query costs below for selected parameter 130 bit security) to highlight our improvement.

4.5 Expanding the Applicability of Fischlin's Transform

As mentioned in the Introduction, Fischlin's transformation applies to only a limited class of Sigma protocols that satisfy a *quasi-unique responses* constraint. Fischlin relied on this property to prove both zero-knowledge as well as proof of knowledge. While it is folklore that this property is not strictly necessary for the extractor, its necessity for zero-knowledge has remained thus far unclear.

We begin by showing in Section 4.5.1 a concrete attack on Witness Indistinguishability when Fischlin's transformation is applied to the Sigma protocol used to prove knowledge

r	Lower bound	This Work	Fischlin
4	1.34×10^{10}	1.34×10^{10}	2.43×10^{10}
5	1.75×10^{8}	1.76×10^{8}	3.36×10^{8}
6	9.97×10^{6}	1.02×10^{7}	2.00×10^{7}
7	1.32×10^{6}	1.40×10^{6}	2.73×10^{6}
8	2.93×10^{5}	3.26×10^{5}	6.23×10^{5}
9	9.25×10^{4}	1.08×10^{5}	2.01×10^{5}
10	3.71×10^{4}	4.55×10^{4}	8.19×10^{4}

Table 4.4.7: Prover work as a function of r for 130-bit security. Fixing the soundness error and the proof size (which is governed by r), this table of analytical estimates shows that our construction almost meets our lower-bound while using a factor of between $2\sqrt{2/\pi}$ and 2 fewer queries than Fischlin's transform.

of one of two discrete logarithms [CDS94]. We then formalize a *strong special soundness* property for Sigma protocols that suffices for extraction, which includes languages that do not by default support the quasi-unique responses property, such as the logical OR Sigma protocol mentioned above. Finally we show how appropriately randomizing Fischlin's construction can achieve ZK unconditionally, for any strong special sound Sigma protocol.

4.5.1 Testing Witness Use in Fischlin's Transformation

Our distinguisher will not rely on the ability to query multiple accepting transcripts for the same challenge. For reference, we first recall the underlying Sigma protocol (due to Cramer et al. [CDS94]) in Figure 4.8.

An adversary attacking Witness Indistinguishability conventionally possesses two witnesses to the theorem and is given a proof π, and must determine which witness was used to produce it. We construct a more powerful type of attack, which makes use of a single witness and determines whether π was created using this witness or the opposite one. This fact will be useful when examining the protocol contexts in which our attack applies.

As we briefly discussed in Section 3.2.2, the attack strategy is to exploit the deterministic nature of Fischlin's prover by retrieving the Sigma protocol randomness and retracing the prover's steps. Concretely with Schnorr-style proofs, the messages z and c and the witness determine the randomness. The attacker can therefore retrieve this randomness, and simply

Protocol $\Sigma^{\vee}_{\mathsf{DL}}$

The prover P and verifier V are both given the statement $(X_0, X_1) = (w_0 \cdot G, w_1 \cdot G) \in \mathbb{G}^2$ while the prover also has $w_b \in \mathbb{Z}_q$ for $b \in \{0, 1\}$.

$\mathsf{P}^{a}_{\Sigma^{\vee}_{\mathsf{DL}}}((X_0, X_1), w_b)$:

1. Simulate a transcript for DLog proof of knowledge of X_{1-b}:
   - Sample $e_{1-b} \leftarrow \{0,1\}^{\kappa}$ and compute $(a_{1-b}, z_{1-b}) \leftarrow \mathcal{S}_{\Sigma_{\mathsf{DL}}}(X_{1-b}, e_{1-b})$
2. Sample $r_b \leftarrow \mathbb{Z}_q$ and compute $a_b = r_b \cdot G$
3. Publish commitment $a = (a_0, a_1)$ and output $\mathsf{state} = w_b, r_b, (a_{1-b}, e_{1-b}, z_{1-b})$

$\mathsf{P}^{z}_{\Sigma^{\vee}_{\mathsf{DL}}}(\mathsf{state}, e)$: Compute $e_b = e \oplus e_{1-b}$ and $z_b = w_b \cdot e_b + r_b$, and Output (e_0, e_1, z_0, z_1)

$\mathsf{V}(X, a, e, z)$:

1. Parse $a = (a_0, a_1)$ and $z = (e_0, e_1, z_0, z_1)$ and verify $e_0 \oplus e_1 = e$
2. Verify $z_b \cdot G = e_b \cdot X_b + a_b$ for each $b \in \{0, 1\}$

Figure 4.8: Proving knowledge of one of two discrete logarithms [CDS94]

replay the honest prover's algorithm and see if the resulting proof string is the same as the given one. The main subtle step in this attack's analysis is to argue that when this retrieve-and-retrace procedure is applied using a different witness from the one used to produce the proof string originally, there is a noticeable probability of producing a different proof string.

While the regular Witness Indistinguishability definition allows the adversary to supply both witnesses, in order to stay within the constraints of quasi-unique responses we formulate a stronger version of the WI experiment for our specific setting. In our definition the challenger samples both witnesses and gives the adversary only one of them (the other witness represents the trapdoor for the system parameter k). We define our experiment as follows:

$\mathsf{Expt}^{\mathsf{DL\text{-}WI}}_{\mathcal{A},\mathsf{P}}(1^{\kappa})$:

1. The adversary $\mathcal{A}$ submits a bit $b \in \{0, 1\}$ to the challenger
2. The challenger samples $w_0, w_1 \leftarrow \mathbb{Z}_q$ and sets $X_0 = g^{w_0}, X_1 = g^{w_1}$
3. The challenger tosses a coin $\beta \leftarrow \{0, 1\}$, and computes $\pi \leftarrow \mathsf{P}((X_0, X_1), w_\beta)$

4. The challenger sends X_0, X_1, w_b, π to $\mathcal{A}$

5. $\mathcal{A}$ outputs a bit

The advantage $\mathsf{AdvDL\text{-}WI}[\mathcal{A}, \mathsf{P}]$ of an adversary $\mathcal{A}$ is defined as:

$$|\Pr[\mathcal{A}(b, w_b, X_{1-b}, \pi) = 1 \mid \beta = 0] - \Pr[\mathcal{A}(b, w_b, X_{1-b}, \pi) = 1 \mid \beta = 1]|$$

Clearly any Witness Indistinguishable scheme will guarantee that the above advantage is negligible. We now give our concrete attack and analysis.

Lemma 4.5.1. *Let* P *be the prover's algorithm obtained by applying Fischlin's transformation [Fis05] to the Sigma protocol to prove knowledge of one of two discrete logarithms [CDS94]. Then there is an efficient adversary* $\mathcal{A}$ *such that* $\mathsf{AdvDL\text{-}WI}[\mathcal{A}, \mathsf{P}]$ *is non-negligible.*

Equipped with this non-negligibly successful adversary $\mathcal{A}$, in Section 4.5.2 we will show how a natural protocol scenario that appears to enable quasi-unique responses in fact structurally resembles the $\mathsf{Expt}^{\mathsf{DL\text{-}WI}}_{\mathcal{A},\mathsf{P}}$ experiment. This allows us to deploy our $\mathsf{Expt}^{\mathsf{DL\text{-}WI}}_{\mathcal{A},\mathsf{P}}$ adversary $\mathcal{A}$ to break the security of the larger protocol.

Proof. For simplicity, we consider only a single base unit, i.e. assume that there is only one repetition in the transformed Sigma protocol.

Consider an attacker, that on input a proof $\pi = ((a_0, a_1), e, (e_0, e_1, z_0, z_1))$ obtained by applying Fischlin's transformation to $\Sigma^{\vee}_{\mathsf{DL}}$ using ℓ-bit output hash function H, and witness w_b, does the following:

1. Compute $r_b = z_b - w_b \cdot e_b$ and set $\mathsf{state}_b = w_b, r_b, (a_{1-b}, e_{1-b}, z_{1-b})$

2. Starting with $e = 0$, increment e until $H((a_0, a_1), e, (e_0, e_1, z_0, z_1)) = 0^\ell$ is found, where $(e_0, e_1, z_0, z_1) = \mathsf{P}^z_{\Sigma^{\vee}_{\mathsf{DL}}}(\mathsf{state}_b, e)$

3. Set $\pi_b = (a_0, a_1), e', (e'_0, e'_1, z'_0, z'_1)$

4. If $\pi_b = \pi$ output b, otherwise output $1 - b$.

Denote the witness used by the challenger to produce the proof as w_β. When $\beta = b$ the attacker outputs the correct bit with certainty since the honest prover's steps are perfectly reconstructed to produce $\pi_b = \pi$. The interesting case to analyze is when $\beta = 1 - b$. There are two possible outcomes triggered in this case, i.e., $\pi_b = \pi$ and $\pi_b \neq \pi$. The latter outcome is induced by the attacker finding an accepting transcript (a, e', z') with $e' < e$ that resulted in $H(a, e', z') = 0^\ell$ (note that $e' > e$ is impossible as we know that $H(a, e, z) = 0^\ell$, and so

the prover never increments past e). The implication in this event is that π was certainly not produced using w_b; this is because had the honest prover started with witness w_b and state state_b, it would have terminated with output $\pi' = (a, e', z')$ rather than the given π.

It remains to show that this distinguishing event (call it $\mathsf{diffProof}$) occurs with non-negligible probability. Note that since the attack is always successful when $\beta = b$, the value $\Pr[\mathsf{diffProof}]$ characterizes the distinguishing advantage of this attack. This is because $\mathsf{AdvDL\text{-}WI}[\mathcal{A}, \mathsf{P}]$ can be simplified as follows, given that b is fixed:

$$|\Pr[\mathcal{A}(w_b, X_{1-b}, \pi) = b \mid \beta = b] - \Pr[\mathcal{A}(w_b, X_{1-b}, \pi) = b \mid \beta = 1 - b]|$$

$$= |1 - (1 - \Pr[\mathsf{diffProof}])| = \Pr[\mathsf{diffProof}]$$

Let $Q_{b,i}$ be the query made by the attacker that corresponds to responding to the i^{th} challenge using witness w_b; in particular

$$Q_{b,i} = (a_0, a_1), i, \mathsf{P}^z_{\Sigma^\vee_{\mathsf{DL}}}(\mathsf{state}_b, i)$$

and thus $\pi_b = Q_{b,i}$ for the smallest i such that $H(Q_{b,i}) = 0^\ell$. Define $Q_{1-b,i}$ the same way using $\mathsf{state}_{1-b} = w_{1-b}, r_{1-b}, (a_b, e_b, z_b)$, except that the query is made by the challenger rather than the attacker in this experiment (since $\beta = 1 - b$).

Claim 4.5.2. *$\forall e' \neq e$, it holds that $Q_{0,e'} \neq Q_{1,e'}$.*

Proof. Consider any $e' \neq e$. Let $e'_0 = e' \oplus e_1$ and $e'_1 = e' \oplus e_0$. Clearly $e'_0 \neq e_0$ and $e'_1 \neq e_1$ as $e' \neq e = e_0 \oplus e_1$. By the structure of $\mathsf{P}^z_{\Sigma^\vee_{\mathsf{DL}}}(\mathsf{state}_b, e')$, the queries $Q_{b,e'}$ are correspondingly constructed as follows:

$$Q_{0,e'} = (\cdots e'_0, e_1, \cdots) \text{ and } Q_{1,e'} = (\cdots e_0, e'_1, \cdots)$$

Clearly $Q_{0,e'} \neq Q_{1,e'}$ as $e_0 \neq e'_0$ and $e_1 \neq e'_1$. □

Corollary 4.5.3. *$\forall e' \neq e$, the values $H(Q_{0,e'})$ and $H(Q_{1,e'})$ are independently distributed.*

Recall that the event $\mathsf{diffProof}$ is precisely the event that the attacker finds an accepting proof $\pi_b = (a, e', z')$ such that $e' < e$. Rather than characterizing $\mathsf{diffProof}$ in its entirety, we analyze a simpler special case. In particular, the event $H(Q_{\beta,0}) \neq 0^\ell$ (implying $e > 0$ in π) and $H(Q_{1-\beta,0}) = 0^\ell$ (implying $e' = 0$ and hence $\pi_b \neq \pi$) induces $\mathsf{diffProof}$. Then applying Corollary 4.5.3 we can therefore lower bound $\Pr[\mathsf{diffProof}]$ as follows:

$$\begin{aligned}
\Pr[\mathsf{diffProof}] &\geq \Pr[H(Q_{\beta,0}) \neq 0^\ell \wedge H(Q_{1-\beta,0}) = 0^\ell] \\
&= \Pr[H(Q_{\beta,0}) \neq 0^\ell] \cdot \Pr[H(Q_{1-\beta,0}) = 0^\ell] \\
&= \frac{2^\ell - 1}{2^\ell} \cdot \frac{1}{2^\ell} = \frac{2^\ell - 1}{2^{2\ell}}
\end{aligned}$$

As we know that $\ell \in O(\log \kappa)$ is necessary for completeness, the denominator of the above value $2^{2\ell} \in \mathsf{poly}(\kappa)$. We therefore conclude that $\Pr[\mathsf{diffProof}]$ is non-negligible in κ, and this completes the analysis.

□

4.5.2 Where to Apply the Attack

Given the attack in Section 4.5.1, when is it safe to apply Fischlin's transformation? Recall that the security of Fischlin's transformation hinges on "quasi-unique responses" as in Definition 3.2.6.

We argue that ensuring this property in a larger system is not always straightforward for languages where the same statement can have multiple witnesses, even when no individual party has more than one witness. In particular, a larger cryptographic application that makes use of such proofs as subprotocols may rely on the ability of the same proof to be produced indistinguishably by different methods, for example by an honest party using a witness in the real protocol, and by a simulator using a trapdoor in the ideal protocol. This subtlety is brought out in the following example protocol between Alice (who only has public input $B \in \mathbb{G}$) and Bob (who has private input $b \in \mathbb{Z}_q$):

- Alice samples $a \leftarrow \mathbb{Z}_q$, sets $A = g^a$, and computes π_A as the Witness Hiding PoK of $\mathsf{DLog}_g(A)$. Alice sends A, π_A to Bob.
- Bob and computes π_B as the WIPoK of $\mathsf{DLog}_g(A) \vee \mathsf{DLog}_g(B)$ using b as a witness. Bob sends B, π_B to Alice.

Fischlin's proof does not directly cover this use case, but it is suggested informally [Fis05, pg. 13] that their construction extends to logical compositions, etc. in the presence of a system parameter enforcing quasi unique responses.

When Alice is corrupt in the above protocol, her view can be simulated without knowledge of b. In particular the simulator simply extracts a from π_A and uses a as a witness to compute π_B as the WIPoK of $\mathsf{DLog}_g(A) \vee \mathsf{DLog}_g(B)$. This simulation is efficient due to extractability of the WHPoK, and will be indistinguishable from the real protocol by witness indistinguishability of the WIPoK. Simultaneously the discrete log b of B is efficiently extractable due to the witness hiding property of WHPoK (in conjunction with the hardness of the discrete logarithm problem) and the extractablity of the WIPoK. This template due to Feige and Shamir [FS90] was used by Pass [Pas03] to construct a deniable two round zero-knowledge argument in the random oracle model, where the simulator does not rely on

programming the random oracle. As shown by Canetti *et al.* [CJS14] this allows for secure composition in the Global Random Oracle model.

Can we safely use Fischlin's transformation here? At first glance, the above protocol appears to be conducive to quasi-unique responses for the sigma protocols that would underlie π_A as well as π_B. Indeed Alice only knows a[4] allowing B to be treated as a system parameter if Alice is corrupt, and Bob only knows b which allows A to be considered a system parameter when Bob is corrupt, therefore neither party has the ability to efficiently compute multiple accepting responses for the same challenge in $\Sigma_{\mathsf{DL}}^{\vee}$.

However this scenario structurally resembles $\mathsf{Expt}_{\mathcal{A},\mathsf{P}}^{\mathsf{DL\text{-}WI}}$, i.e. since our attack on WI for Fischlin's transformation does not require knowledge of both witnesses, it can be applied here. In particular Alice knows a, and so can test whether the proof π_B was computed using a or b as the witness. This allows her to distinguish between the real protocol (where Bob uses b as the witness to compute π_B) and the simulation (where π_B is generated by the simulator using a as the witness).

We therefore make the case that a cleaner definition is required, ideally one that does not require reasoning about the context in which a sigma protocol is used.

4.5.3 Strong Special Soundness

Before describing how to patch the above attack, we present an easily verifiable property of Sigma protocols for which our transformation applies. Rather than attempting to quantify the ability of an adversary to induce a bad event, we take a constructive approach in our definition; i.e., it is easier to evaluate precise deterministic conditions (such as special soundness) rather than reason about probabilistic/computational system parameters (as in quasi-unique responses).

Our definition is a mild strengthening of the two-special soundness notion for Sigma protocols [Dam02], and so we call it *strong* two-special soundness—also in homage to the similar concept of *strong* unforgeability for signature schemes. Informally stated, a strongly two-special sound sigma protocol has an extractor which when given two distinct accepting transcripts (a, e, z) and (a, e', z') that share the same first message, outputs a witness for the statement with certainty (note that $e = e'$ is allowed). The standard two-special soundness notion enforces that $e \neq e'$ for the extractor's success. We give the formal definition in Definition 2.5.2 in Section 2.5.

[4]We ignore the prospect of obtaining auxiliary information about b, for eg. b could be sampled uniformly as part of a larger protocol.

Many natural sigma protocols (including logical compositions [CDS94], Okamoto's identification protocol [Oka93], etc.) satisfy this definition (but may not satisfy quasi-unique responses). There are two notable natural examples that may not meet this definition: (1) Blum's protocol to prove knowledge of a Hamiltonian cycle [Blu86] allows the prover to open any cycle in the graph and it is unclear as to how an extractor for strong special soundness can deal with such a situation, and (2) the Sigma protocol that underlies EdDSA [BDL+12], which is Schnorr's scheme implemented over an elliptic curve group of composite order. The lax verification equation in the original specification means that the verifier accepts multiple discrete logarithms for the same curve point. However we stress that this is due to lax realization of the abstraction required for Schnorr's sigma protocol, and is easily fixed in works that succeeded the original spec [CGN20, BCJZ21]. Note that both cases will not support quasi-unique responses either, if they are not strong special sound.

Note that any standard Sigma protocol that is not strongly two-special sound can not have quasi-unique responses. In particular by definition the only way to retain standard special soundness while violating strong two-special soundness is by presenting accepting transcripts $(a, e, z_1), (a, e, z_2)$ that do not yield a witness for the theorem when given to the extractor. Any notion of *efficient* adversaries being unable to find such transcripts in the case of quasi-unique responses is captured by amending the theorem for the strong two-special sound Sigma protocol to include a disjunctive clause for knowledge of the system parameter trapdoor.

With our definition in place, we study how to compile such Sigma protocols to NIZKPoKs using Fischlin's technique.

4.5.4 Randomization Extends Fischlin's Technique

The issue in Fischlin's transformation is that the prover's algorithm is deterministic and consequently re-traceable. Indeed, if one were to instantiate the transformation of Pass [Pas03] by simply constructing a hash tree of accepting protocol transcripts instead of a Merkle tree of *commitments* to such transcripts, the same issue as described above would present itself more directly: given a proof and candidate witness for the statement, one could simply extract the prover's randomness and test if recomputing the proof once again yields the given one. This issue is implicitly avoided by Pass (at constant factor overhead) by constructing the Merkle tree with commitments to protocol transcripts. However it is unclear how to make such an approach work with Fischlin's transform; using randomized commitments appears to be at odds with obtaining soundness.

We show that an alternate method of randomization can be used to extend Fischlin's

technique to any strong special sound Sigma protocol. The idea is to randomize the NIZK prover's algorithm so that the prover randomly steps through the challenge space until an accepting transcript that hashes to the all-zero string is found. Intuitively, proofs produced with this modified transformation do not leak any information about how many queries the prover had to make in order to find an accepting transcript. This makes it impossible for a distinguisher to retrace the steps of a prover even given all witnesses as it does not have access to the random sequence in which the prover queried the random oracle. We give a formal description of the modified transformation in Figure 4.9 below, along with a proof of security.

Protocol $\pi_{\mathsf{NIZK}}^{\mathsf{F\text{-}rand}}$

The prover P and verifier V are both given the statement x while the prover also has a witness w for the statement $x \in L$. The security parameter κ defines the integers r, ℓ, t. These integers are related as $r \cdot \ell = 2^\kappa$, and $t = \lceil \log \kappa \rceil \cdot \ell$. Both parties have access to a Random Oracle $H : \{0,1\}^* \mapsto \{0,1\}^\ell$. The underlying sigma protocol is given by $\Sigma = ((\mathsf{P}_\Sigma^a, \mathsf{P}_\Sigma^z), \mathsf{V}_\Sigma)$.

$\mathsf{P}^H(x, w)$:

1. For each $i \in [r]$, compute $(a_i, \mathsf{state}_i) \leftarrow \mathsf{P}_\Sigma^a(x, w)$
2. Set $\boldsymbol{a} = (a_i)_{i \in [r]}$
3. For each $i \in [r]$, do the following:
   (a) Set $\mathcal{E}_i = \emptyset$
   (b) Sample $e_i \leftarrow \{0,1\}^t \setminus \mathcal{E}_i$ and compute $z_i = \mathsf{P}_\Sigma^z(\mathsf{state}_i, e_i)$
   (c) If $H(\boldsymbol{a}, i, e_i, z_i) \neq 0^\ell$, update $\mathcal{E}_i = \mathcal{E}_i \cup \{e_i\}$ and repeat Step 3b
4. Output $\pi = (a_i, e_i, z_i)_{i \in [r]}$

$\mathsf{V}^H(x, \pi)$:

1. Parse $(a_i, e_i, z_i)_{i \in [r]} = \pi$, and set $\boldsymbol{a} = (a_i)_{i \in [r]}$
2. For each $i \in [r]$, verify that $H(\boldsymbol{a}, i, e_i, z_1) = 0^\ell$ and $\mathsf{V}_\Sigma(x, (a_i, e_i, z_i)) = 1$, aborting with output 0 if not
3. Accept by outputting 1

Figure 4.9: Randomized Fischlin's Transformation

Extractor $\mathsf{Ext}_{\mathsf{NIZK}}$

The extractor is given the statement x, a proof π, and the list of queries to the random oracle $\boldsymbol{Q}$ that were made by the adversary in the production of this proof. In addition to this, this extractor has access to the extractor Ext_{Σ} of the strongly special sound sigma protocol, which requires 2 accepting transcripts (with the same a value) in order to produce a witness w for the statement.

$\mathsf{Ext}_{\mathsf{NIZK}}(x, \pi, \boldsymbol{Q})$:

1. Parse $(a_i, e_i, z_i)_{i\in[r]} = \pi$, and set $\boldsymbol{a} = (a_i)_{i\in[r]}$
2. Search $\boldsymbol{Q}$ until a query of the form $(\boldsymbol{a}, i, e, z)$ is found such that $(e, z) \neq (e_i, z_i)$, and $\mathsf{V}_{\Sigma}(x, a_i, e, z)) = 1$
3. Output $\mathsf{Ext}_{\Sigma}(a_i, e_i, e, z_i, z)$

Figure 4.10: Extracting a witness

Theorem 4.5.4. *If Σ is a strongly two-special sound sigma protocol for the language L, then protocol $\pi_{\mathsf{NIZK}}^{\mathsf{F\text{-}rand}}$ is a straight-line extractable non-interactive zero-knowledge proof of knowledge for the language L in the random oracle model.*

Proof. **Completeness:** follows from the same analysis as Fischlin [Fis05]. Denote by Q_{i,e_i} the query made by P in Step 3c of its algorithm. The only event in which the prover does not find an accepting proof is when $\exists i \in [r]$ such that $\forall e_i \in \{0,1\}^t$, $H(Q_{i,e_i}) \neq 0^\ell$. Call this event fail. As each $H(Q_{i,e_i})$ is independent, we can bound the probability of fail as follows:

$$\begin{aligned}
\Pr[\mathsf{fail}] &= \Pr[\exists i \in [r] : \forall e_i \in \{0,1\}^t,\ H(Q_{i,e_i}) \neq 0^\ell] \\
&\leq \sum_{i\in[r]} \Pr[\forall e_i \in \{0,1\}^t,\ H(Q_{i,e_i}) \neq 0^\ell] \\
&= \sum_{i\in[r]} \prod_{e_i\in\{0,1\}^t} \Pr[H(Q_{i,e_i}) \neq 0^\ell] \\
&= \sum_{i\in[r]} \prod_{e_i\in\{0,1\}^t} \left(1 - \frac{1}{2^\ell}\right) = r \cdot \left(1 - \frac{1}{2^\ell}\right)^{2^t} \\
&= r \cdot \left(1 - \frac{1}{2^\ell}\right)^{\kappa\cdot 2^\ell} \approx r \cdot \frac{1}{e^\kappa} \\
&\leq 2^{-\kappa}
\end{aligned}$$

Proof of knowledge: This follows from the same analysis as Fischlin [Fis05] as well.

The event in which this extractor fails is the event in which an adversarial prover P^* is able to produce a proof π by querying no more than a single accepting Sigma protocol

Simulator $\mathcal{S}^{\mathsf{F}}_{\mathsf{NIZK}}$

Simulator $\mathcal{S}^{\mathsf{F}}_{\mathsf{NIZK}}$ is given the statement x, and has the ability to program the Random Oracle H. In addition to this $\mathcal{S}^{\mathsf{F}}_{\mathsf{NIZK}}$ is given the simulator for the Sigma protoocol $\mathcal{S}_{\Sigma}$.

$\mathcal{S}^{\mathsf{F}}_{\mathsf{NIZK}}(x)$:

1. Uniformly sample $e_i \leftarrow \{0,1\}^t$ for each $i \in [r]$ and set $\boldsymbol{e} = (e_i)_{i\in[r]}$
2. Run the simulator for the sigma protocol to obtain $(a_i, z_i) \leftarrow \mathcal{S}_{\Sigma}(x, e_i)$ for each $i \in [r]$
3. Program the random oracle H so that $H(\boldsymbol{a}, e_i, z_i) = 0$ for each $i \in [r]$
4. Emulate H as a random oracle 'honestly' for every other query
5. Output $\pi = (\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z})$

Figure 4.11: Simulator for Zero-Knowledge

transcript for each $i \in [r]$ to the random oracle. We first ignore all queries made to H that are not accepting transcripts, and then separate queries prefixed by different $\boldsymbol{a}$ as they essentially instantiate independent random oracles (and can not be combined with one another to produce a proof). For a given $\boldsymbol{a}$, the event in which the adversary is able to output an accepting proof with fewer than 2 accepting transcripts (prefixed by $\boldsymbol{a}$) queried to H for each $i \in [r]$ is exactly the event that the first such accepting transcript queried to H for every $i \in [r]$ evaluates to 0. This is equivalent to r independent uniformly chosen ℓ-bit strings being equal to 0, which happens with probability $(2^{-\ell})^r = 2^{-\kappa}$. For an adversary that makes $|\boldsymbol{Q}|$ queries to the random oracle, the extraction error is therefore bounded by $|\boldsymbol{Q}|/2^{\kappa}$.

Zero-knowledge: We describe how to simulate a proof in Figure 4.11, and then show its indistinguishability from a real proof.

We argue that the simulation is indistinguishable from a real proof through a sequence of hybrid experiments, which are defined as follows.

Hybrid $\mathcal{H}_1$. The real prover's algorithm (P from $\pi^{\mathsf{F\text{-}rand}}_{\mathsf{NIZK}}$) is used to find $(\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z})$ such that $H(\boldsymbol{a}, e_1, z_1) = \cdots = H(\boldsymbol{a}, e_r, z_r) = 0$ where H is emulated as a random oracle by the standard technique of maintaining a (query, response) table. The difference from the real prover's algorithm is merely syntactic.

Hybrid $\mathcal{H}_2$. Implement Step 3 of $\mathcal{S}^{\mathsf{F}}_{\mathsf{NIZK}}$. In particular in this experiment, the random oracle H is implemented as follows:

1. The first r queries by the honest prover $Q_1, Q_2, \cdots Q_r$ (where each $Q_i = (\boldsymbol{a}, e_i, z_i)$ as generated by P) will receive 0 as a response, i.e. $H(Q_1) = H(Q_2) = \cdots = H(Q_k) = 0$

2. Emulate H as a random oracle 'honestly' for every other query

This hybrid differs from the last in that here the prover P will terminate after the first r queries it makes to H, whereas in $\mathcal{H}_1$ since H is not programmed to shortcut to 0, P will have to 'work' to find accepting transcripts that evaluate to 0. Since the difference in running time of $\mathcal{H}_2$ and $\mathcal{H}_1$ is invisible to a distinguisher and $\boldsymbol{a}$ are generated identically in both hybrids, the only component that remains to be analyzed is $\boldsymbol{e}$ (since $\boldsymbol{z}$ is implicitly fixed by $w, \boldsymbol{a}, \boldsymbol{e}$). In $\mathcal{H}_1$, each e_i represents the index at which the first pre-image of 0 was found by P relative to $H(\boldsymbol{a}, i, \cdot)$. Since P steps through pre-images uniformly at random and H is a random oracle (i.e. H has independent uniformly random outputs for every pair of distinct inputs) each e_i is distributed uniformly in $\{0,1\}^t$ in $\mathcal{H}_1$. In $\mathcal{H}_2$, each e_i is clearly uniformly distributed in $\{0,1\}^t$ as it corresponds to the first r challenges tried by P, which are sampled uniformly and independently.

As $\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z}$ are distributed identically in $\mathcal{H}_2$ and $\mathcal{H}_1$, the only distinguishing event corresponds to the programming of H, i.e. if the adversary is able to query H on some index that $\mathcal{H}_2$ subsequently programs to a different value. Since $\boldsymbol{a}$ has at least κ bits of entropy and is a prefix for all queries programmed in $\mathcal{H}_2$, this distinguishing event happens with probability no greater than $|\boldsymbol{Q}|/2^\kappa$, where $|\boldsymbol{Q}|$ is the number of queries made by the adversary to the random oracle.

Hybrid $\mathcal{H}_3$. We define hybrid experiment ${\mathcal{H}_3}^0$ to be the same as the last, with the only change being that the vector of challenges $\boldsymbol{e}$ is sampled before invoking $P_{\Sigma,a}$. This change is merely syntactic, and ${\mathcal{H}_3}^0$ is distributed identically to $\mathcal{H}_2$. We now define a sequence of sub-hybrids $\{{\mathcal{H}_3}^i\}_{i\in[r]}$ as follows: hybrid experiments ${\mathcal{H}_3}^{i-1}$ and ${\mathcal{H}_3}^i$ are identical except that they differ in their computation of (a_i, z_i). In particular, ${\mathcal{H}_3}^{i-1}$ computes $(a_i, \mathsf{state}_i) \leftarrow P_{\Sigma,a}$ and $z_i \leftarrow P_{\Sigma,z}(e_i, \mathsf{state}_i)$ whereas ${\mathcal{H}_3}^i$ computes $(a_i, z_i) \leftarrow \mathcal{S}_\Sigma(x, e_i)$. Clearly distinguishing ${\mathcal{H}_3}^{i-1}$ from ${\mathcal{H}_3}^i$ is equivalent to distinguishing a simulated Sigma protocol transcript from a real one. By perfect simulation of the sigma protocol, we have that ${\mathcal{H}_3}^{i-1} \equiv {\mathcal{H}_3}^i$ for each $i \in [r]$.

The final experiment in this sequence ${\mathcal{H}_3}^r$ implements Steps 1 and 2 of $\mathcal{S}^{\mathsf{F}}_{\mathsf{NIZK}}$ and is entirely independent of the witness, which completes the process of replacing the real $\mathsf{P}(x, w)$ with the simulation $\mathcal{S}^{\mathsf{F}}_{\mathsf{NIZK}}(x)$.

□

4.6 Open Problems

In this chapter, we showed how to aggregate Schnorr signatures by a factor of 2, with techniques that are blackbox in the hash function, and proved that this compression rate is optimal. It remains open to construct non-blackbox techniques in hash functions that enable better aggregation methods than invoking generic SNARKs.

We showed how to speed up aggregation when tight security is desired, so that the overhead of tight security is roughly three orders of magnitude over the non-tight version, for real-world parameters. This overhead is already tolerable for many applications, but of course could have room for improvement. We showed how these techniques can be of benefit to the zero-knowledge setting as well. We were able to prove a lower bound on Prover query complexity in the zero-knowledge setting, and showed that Fischlin's construction is roughly a factor of 2–2.7 worse than this bound. We showed (for the first time) how to achieve optimal prover query complexity for a small range of parameters; an interesting question for future work is whether the bound can be met (or improved) for larger parameter ranges. Of course, Fischlin showed that such a bound can be circumvented when the straight-line extractor is allowed to program the random oracle, but thus far no meaningfully efficient techniques to circumvent the bound are known.

Finally, we showed that Fischlin's transformation is insecure when applied to certain common Sigma protocols, such as the proof of knowledge of one of two discrete logarithms. We attributed this flaw to the deterministic nature of the transform, and showed how suitable randomization can fix the problem. An interesting open problem is whether any "well-behaved" compiler that achieves straight-line extraction must necessarily be randomized.

Chapter 5

Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions

The fact that r values are chosen uniformly upon every invocation of Schnorr's signing algorithm permits many useful theoretical properties, among them a clean proof [PS96]. However, the necessity of fresh randomness introduces a new attack vector in practice: the assumption that a consistent source of entropy will be available for use has repeatedly turned out to be ill-founded.

As an example, the public cloud is a context in which access to good entropy and a well-seeded PRNG is particularly difficult. Indeed, deploying an application on cloud infrastructure delivers the convenience of modern enterprise-grade offerings, yielding significant benefits in uptime, availability, APIs, threat detection, load balancing, storage, and more. Yet this choice often entails that a user application will run as a guest in a virtualized environment of some kind. Existing literature shows that such guests have a lower rate of acquiring entropy [KASN15], that their PRNG behaves deterministically on boot and reset [EZJ+14], and that they show coupled entropy in multi-tenancy situations [KC12], including in containers [Bay14]. This can have disastrous consequences, as even a small amount of bias in r values across many Schnorr signatures can be leveraged to completely break the scheme [HS01, ANT+20, MH20].

5.1 Stateless Deterministic Signing

The idea of deterministically deriving r values from randomness established during key generation has correspondingly gained traction [MNPV99, KW03, KL17]. The widely used EdDSA signature scheme [BDL+12] derives its nonces as $r = H(H(k), m)$ where k is sampled during key generation and m is the message to be signed. Assuming k has enough entropy and that H produces pseudorandom outputs, the r values will be pseudorandomly determined for each m, leading to signatures that are essentially as secure as the original Schnorr algorithm that consumes fresh randomness for each r. In this chapter, we aim to translate the benefits of deterministic signing to the threshold signature setting. In particular, we study *deterministic threshold Schnorr* as the problem of designing a decentralized protocol to produce Schnorr signatures where each party's signing algorithm is deterministic.

State Continuity is non-trivial. Folklore would suggest that the problem at hand is simple: first design a randomized protocol (of which many exist for threshold Schnorr) and then simply 'compress' the random tape by using a PRG/block cipher invoked with a fresh counter each time new randomness is needed. However this approach fundamentally assumes *state continuity* [PLD+11], i.e. that the state of the device running the protocol can be reliably updated and read on demand. However as Parno et al. [PLD+11] first pointed out, even secure hardened devices with strong isolation guarantees can not take this property for granted. In particular, malicious attackers or even natural circumstances such as software errors or power interruptions may induce a device to turn off and roll back to a 'last known safe' state upon restart. While such a state may be entirely consistent in every detectable way, it could be stale, leading to randomness reuse in the PRG context. We stress that reliably storing long-term secrets is significantly easier than for instance updating a counter every time a signature is produced.

Why not solve this at the systems level? While state continuity in general has been studied as a systems problem, we argue here that incorporating resiliency to state resets in cryptographic protocol design has both qualitative and quantitative advantages:

- **Qualitative:** Systems-level solutions depend on context, and consequently hinge on specific assumptions such as trusted hardware [PLD+11, SP16], a number of helper nodes with uncorrelated failures [BCLK17, MAK+17], or a trusted server [vDRSD07]. In contrast, a cryptographic protocol in the standard model offers provable security and strong composition guarantees without resorting to context-specific physical assumptions.

- **Quantitative:** Deployment of a protocol that relies on state continuity will require the expending of resources on establishing context-specific solutions to this problem for each new environment; acquiring such dedicated hardware is expensive. Moreover it is unclear that the best systems solution in every environment will be more efficient than a canonical stateless cryptographic protocol. Consider two-party distributed signing: it defeats the purpose to incorporate extra parties/servers for state continuity, and solutions that rely on monotonic counters maintained on special purpose hardware such as Intel SGX or Trusted Platform Modules suffer other issues inherent to these platforms. Matetic et al. [MAK+17] showed that the upper limit on the number of writes to such protected counters (due to non-volatile memory wearing out) can be exhausted in a few days of continuous use. Moreover maintaining and reading from such memory is slow; Strackx and Piessens [SP16] report a 95ms latency, and Brandenburger et al. [BCLK17] report a 60ms latency in incrementing an SGX Trusted Monotonic Counter. In summary, dedicated hardware for state continuity is expensive, slow, and comes with limited lifespan. The protocols we construct in this work are expected to run significantly faster on commodity hardware (order of 10ms) - well within the performance envelope of trusted hardware solutions due to their latency.

We therefore incorporate *statelessness* into the problem statement, to mean that security must hold even when devices are arbitrarily crashed (even in the middle of a protocol) and restored with only their long-term secrets.

5.1.1 Why is Stateless Deterministic Threshold Signing Challenging?

Schnorr signatures permit a very elegant multiparty variant [SS01, GJKR07] as signatures are simply linear combinations of secret values. Most natural secret sharing schemes permit linear combinations "for free", with the result that threshold Schnorr in different settings has essentially reduced to the task of establishing $x \cdot G$ and $r \cdot G$ such that x, r are random and secret-shared among parties.

The instructions for each of the parties in the classic threshold Schnorr protocols [SS01, GJKR07] looks very much like the regular signing algorithm. We give a brief sketch of how the semi-honest two party version works, which is sufficient to understand the challenges we address in the rest of our exposition. The description is from the point of view of party P_b for $b \in \{0, 1\}$.

1. **Key generation**: P_b samples $\mathsf{sk}_b \leftarrow \mathbb{Z}_q$ and sets $\mathsf{pk}_b = \mathsf{sk}_b \cdot G$. It then sends pk_b to P_{1-b} and waits for pk_{1-b}. The shared public key is set to $\mathsf{pk} = \mathsf{pk}_0 + \mathsf{pk}_1$

2. **Given message m to sign**:

   (a) P_b samples $r_b \leftarrow \mathbb{Z}_q$ and sets $R_b = r_b \cdot G$. It then sends R_b to P_{1-b} and waits for R_{1-b}. The signing nonce is computed by both as $R = R_0 + R_1$

   (b) P_b computes $e = H(\mathsf{pk}, R, m)$ and sets $\sigma_b = \mathsf{sk}_b \cdot e + r_b$. It then sends σ_b to P_{1-b} and waits for σ_{1-b}. Finally $(R, \sigma = \sigma_0 + \sigma_1)$ is a signature on m

The above protocol can be made secure against an active adversary with an extra commitment round [NKDM03], however this will not be important for our discussion. An immediate observation is that instead of having P_b sample a fresh r_b for each message, one could adopt the EdDSA approach and have P_b sample k_b during key generation and instead compute $r_b = H(H(k_b), m)$. This does in fact yield a deterministic threshold signing protocol, with security against at least passive corruption. However, as previously noted by Maxwell et al. [MPSW19], a malicious adversary will be able to completely break such a system. The attack is as follows: a malicious P_1 can first run the honest signing procedure for message m with the correct r_1 (as per Step 2a), and subsequently ask to sign the same m but use a different $r_1' \neq r_1$ this time. The honest P_0 follows the protocol specification and unfortunately uses the same r_0 value in both executions, as it is derived as a function of m, k_0, both of which are independent of r_1. Consequently, $R = (r_0 + r_1)G$ and $R' = (r_0 + r_1')G$ are the nonces derived for each execution, which induce unequal challenges $e = H(..R)$ and $e' = H(..R')$. The honest party therefore gives P_1 the values $\sigma_0 = \mathsf{sk}_0 e + r_0$ and $\sigma_0' = \mathsf{sk}_0 e' + r_0$ in different executions, which jointly reveal its secret key share sk_0.

Going forward, we follow the natural template for threshold Schnorr set by previous works [NKDM03, SS01, GJKR07], and investigate how to enforce that parties indeed derive their nonces deterministically by following the protocol.

5.1.2 Desiderata

There are many possible ways to enforce deterministic nonce derivation, and so we first highlight the constraints of our context in order to inform our choice of technology.

- **Standard assumptions.** This is a subtle but important constraint. As with any safety critical application, we would like to avoid new and insufficiently vetted assumptions in our protocol. However even more important than protocol security (which is only relevant when parties in the system are corrupt) is the security of artefacts exposed to the outside world, i.e. the signature. In particular, we wish to be very conservative in instantiating the PRF that is used to derive nonces; Schnorr signatures are known to be extremely sensitive

to nonce bias [HS01, TTA18], meaning that the slightest weakness discovered in the PRF could lead to attackers retrieving entire signing keys using previously published signatures.

- **Lightweight computation.** We want our schemes to be as widely applicable as possible, and consequently we do not want to make use of heavy cryptography. In the case of decentralized cryptocurrency wallets where one or more signing party is likely to be a weak device (e.g. low budget smartphone, or Hardware Security Module) both computation cost and memory consumption must be minimized. On the other end of the spectrum for threshold signing at an institutional level with powerful hardware, lightweight signing is conducive to high throughput.

- **Round efficiency.** As much as possible we would like to avoid compromising on round efficiency in our endeavour to make signing deterministic and stateless. In particular ordinary threshold Schnorr signing [NKDM03, SS01, GJKR07] requires only three rounds, and we would like to match this efficiency.

We therefore formulate a more precise problem statement,

> How can we construct a lightweight threshold signing protocol for Schnorr signatures where parties do not consume fresh randomness or update state after the initial key generation? Moreover nonce derivation must strictly use standardized primitives (eg. AES, SHA).

To be clear, our focus is on the 'online' signing operations; we do not worry about optimizing the efficiency of the distributed key generation, which is one-time.

5.1.3 This Work

In this work, we construct an efficient zero-knowledge proof system for proving correct nonce derivation that makes use of only cheap symmetric key cryptography and a small constant number of exponentiations when invoked, and does not require updating long-term state.

Our proof system is in the Zero-knowledge from Garbled Circuits (ZKGC) paradigm of Jawurek et al. [JKO13], and the techniques that we develop improve the ZKGC paradigm even outside of the stateless deterministic setting.

General ZKGC Bottlenecks. The efficiency of the ZKGC paradigm is rooted in the fact that the prover and verifier pay at most three AES invocations per AND gate in the circuit, when instantiated with the privacy-free variant of HalfGates [ZRE15]. However especially for small circuits such as AES, SHA, etc., the bottleneck usually lies in logistics for the witness (i.e. input to the circuit). In particular:

- **Input Encoding:** Transferring wire labels for a $|q|$-bit input requires $|q|$ Oblivious Transfers, which means $O(\kappa)$ public key operations per invocation even with OT Extension, for κ bits of computational security.
- **Binding Composite Statements:** The state of the art technique [CGM16] to tie statements about an order q elliptic curve group elements to a Boolean circuit involves the garbling of an additional private circuit to multiply a $|q|$-bit value with an s-bit statistical MAC. While the cost of these $\tilde{O}(s \cdot |q|)$ extra gates may disappear as the circuit size grows, it incurs high concrete cost relative to common ciphers that have compact Boolean circuit representations. Consider the parameter regime relevant here, a 256-bit curve and 60 bits of statistical security: the cost of garbling with privacy ($2\times$ the per-gate cost of privacy-free [ZRE15]) the corresponding 32k gate multiplication circuit for the MAC[1] is considerably more expensive than privacy-free garbling of the circuit for the relation itself: nearly an order of magnitude more than AES-128 (7k gates [AMM+]) and even $3\times$ that of SHA-256 (22k gates [CGGN17]).

We develop novel techniques to address both of these problems in this work, which we briefly describe below.

Commit Once, Prove Many Statements

As the use of $O(\kappa)$ public key operations used for input encoding in garbled circuit based protocols is a difficult foundational issue, we relax the problem to fit our setting more closely. In particular, it is sufficient for a party to commit to a nonce derivation key k once during distributed key generation, and subsequently prove an unbounded number of statements (i.e. PRF evaluations) online. This gives us a more targeted problem statement:

> How can we enable the prover to commit to its witness w once, and prove an unbounded number of statements x such that $R(x, w) = 1$ with only symmetric key operations per instance in the ZKGC paradigm?

This problem reduces to the task of constructing a variant of Committed OT, where an OT receiver commits to a choice bit once and subsequently receives one out of two messages for an unbounded number of message pairs sent by the sender. Importantly, after the sender has sent a message pair, the sender should be able to reveal the pair at a later point without changing the messages or learning the receiver's choice bit. We devise a novel method that makes non-blackbox use of any Universally Composable (UC) commitment [Can01] in the

[1] Calculated with Karatsuba's multiplication algorithm per Table 6.7 in [CCD+20]

one-time preprocessing model to solve this problem. Roughly, each OT in the canonical instantiation of input encoding is replaced by a pair of UC commitments. This is substantially more computationally efficient, as we summarized in Table 3.3.2 in Section 3.3.

On a Macbook Pro 2017 laptop (i7-7700HQ CPU, 2.80GHz) running OpenSSL 1.1.1f: a single AES-128 invocation takes $0.07\mu s$, SHA-512 takes $0.3\mu s$, and a Curve25519 exponentiation takes $59.8\mu s$. This data in combination with Table 3.3.2 suggests that our technique for preprocessing Committed OT can perform input encoding an order of magnitude faster than using the fastest plain OT.

Beyond Stateless Deterministic Signing. This pattern of proving an unbounded number of statements about the same private input is not unique to threshold signing. Consider the example of distributed symmetric key encryption [AMMR18]: Servers A and B (one of which may be malicious) hold keys k_A, k_B respectively, and comprise one endpoint of a secure channel. Ciphertexts on this channel are of the form $(r, m \oplus \mathsf{F}_{k_A}(r) \oplus \mathsf{F}_{k_B}(r))$, and so encryption/decryption requires the servers to reveal $\mathsf{F}_{k_A}(r), \mathsf{F}_{k_B}(r)$ and prove correct evaluation.

Exponentiation Garbling Gadget

We design a gadget to garble the exponentiation function $f_G(x) = x \cdot G$ at very low cost. The gadget takes as input a standard Yao's garbled circuit style encoding of a bit string $\mathbf{x}$ (i.e. keys $(k_i^{\mathbf{x}_i})_{i \in [|\mathbf{x}|]}$), and outputs a convenient algebraic encoding of this value $Z = (ax + b) \cdot G$ for some secret $a, b \in \mathbb{Z}_q^*$.

A similar effect is achieved by Chase et al. [CGM16] by garbling an explicit multiplication circuit. However our gadget is drastically more efficient, as summarized in Table 3.3.1 in Section 3.3.

This leads to significant savings, as stated earlier the MAC computation alone would have dominated bandwidth cost.

Beyond Stateless Deterministic Signing. This gadget cuts down the heavy MAC computation in [CGM16] by a factor of 125, and therefore is useful for composite statements where the Boolean circuit size for the non-algebraic component is smaller or comparable in size to $\tilde{O}(s \cdot |q|)$. Concretely bandwidth savings can range from $\sim$ 90% for AES-128, to $\sim$ 70% for SHA-256. The latter translates significant bandwidth savings in the context of proving knowledge of an ECDSA signature [CGM16].

We are therefore able to construct a highly efficient zero-knowledge proof system, where a prover commits to a some nonce derivation key k during key generation, and subsequently proves correct nonce derivation, i.e. $R = \mathsf{F}_k(m) \cdot G$ for an unbounded number of messages

m that are signed online. Simply augmenting the semi-honest threshold signing protocol sketched earlier with this zero-knowledge proof yields an n-party stateless deterministic threshold signing protocol that is secure against $n-1$ malicious corruptions.

5.2 Related Work

Resettable Zero-knowledge (rZK). The notion of rZK introduced by Canetti et al. [CGGM00] allows an adversarial verifier to arbitrarily reset a prover, and requires zero-knowledge to hold even in the absence of fresh randomness for the prover upon being reset. This achieves stateless determinism as we require, and indeed the attacks discovered by Canetti et al. on canonical protocols when confronted with such an adversary are of the same flavour as the one in Section 5.1.1. However the adversarial model that we consider in this work is weaker for two reasons: one is that the prover and verifier are allowed a one-time reset-free interactive setup phase, and the other is that in case an abort is induced at any point no further interaction will occur. Therefore rZK protocols would be overkill for our setting.

MuSig-DN. The closest work to ours is the very recent work of Nick et al. [NRSW20], in which the authors construct a two-round multisignature scheme called MuSig-DN which enforces deterministic nonces with security against $n-1$ out of n malicious corruptions. Their protocol achieves stateless deterministic signing for Schnorr signatures, however their approach diverges from ours in two significant ways:

- The security of the PRF they use for nonce derivation is based on the Decisional Diffie-Hellman assumption over a carefully chosen custom elliptic curve that supports efficient proofs. While this offers a nice tradeoff between the efficiency of proving statements about arithmetization-friendly primitives and plausibility of assumptions, the assumption is not exactly the same as DDH over a standardized curve.
- They opt for a SNARK-based approach (specifically Bulletproofs [BBB+18]), which is very communication efficient (around a kilobyte for a proof) but computation intensive; they report 943ms on commodity hardware for a single execution.

In contrast, our dishonest majority protocol occupies a different point on the spectrum: it supports standardized ciphers for nonce derivation, and is computationally very light at the expense of higher bandwidth.

Threshold EdDSA. Due to the fact that the EdDSA signing algorithm derives r as a non-linear function of some preprocessed seed, securely computing EdDSA in a threshold setting exactly as per its specification is quite challenging. Current implementations of threshold EdDSA either require elaborate (randomized) MPC protocols [BST21] or abandon deterministic nonce derivation altogether and simply implement randomized threshold Schnorr over the correct curve [LPR19]. As an example, the Unbound library [LPR19] drops the determinism requirement with the justification that a nonce jointly sampled in a multiparty protocol will be uniformly random if even one of the parties uses good randomness. However this does not protect a device using bad randomness from a malicious adversary controlling a party in the system. Moreover we contend that in practice it is common for all parties in the system to be using similar implementations, hence inducing correlated randomness-related vulnerabilities. Additionally faults/bugs may occur at the system or hardware levels, which further motivates the need for threshold signing protocols that do not assume that *any* party in the system has reliable randomness.

In this work we are not concerned with *exactly* computing the correct EdDSA signing equation in a distributed setting, as this will likely require expensive MPC [BST21]. Instead we would like to construct a threshold Schnorr protocol that embodies the spirit of deterministic nonce derivation; in particular our primary goal is to construct a multiparty protocol to compute Schnorr signatures where *each particpant* runs a deterministic and stateless signing algorithm. Also note that the work of Bonte et al. [BST21] is in the incomparable honest majority setting, and highly interactive.

5.3 Our Techniques

The task at hand can be roughly characterized as follows: parties in the system first sample some state during a "key generation" phase. When given a message to sign later, they must securely derive the signing material from the joint state they sampled earlier. Moreover, this derivation must be deterministic and should not create new state, i.e. signing material for each message must only rely on the key generation state and the message itself. The template of sampling a PRF key during key generation and applying this PRF on the message to be signed to derive signing material works well in the semi-honest setting as discussed, but falls apart when adversaries deviate from the protocol.

The canonical method to upgrade a semi-honest protocol to malicious security without an honest majority is for parties to commit to some initial randomness, and subsequently prove that they computed each message honestly relative to the committed randomness [GMW87]. What this entails for threshold Schnorr is for parties to commit to a PRF key during dis-

tributed key generation, and when signing a message, prove that the discrete log of their claimed nonce is indeed the result of applying the PRF on the public message using the committed key. In particular for some public $x, R_i, \mathsf{Commit}(k_i)$, party P_i must prove that $\mathsf{F}_{k_i}(x) \cdot G = R_i$ where F is a PRF. We encapsulate this mechanism in the functionality $\mathcal{F}_{\mathsf{F},\mathbb{G}}$ later in this paper.

5.3.1 What existing proof technologies suit our task?

As per our desiderata that we set in Section 5.1.2, we wish to prioritize *standard assumptions*, *light computation*, and *retaining round efficiency*. We examine the different proof technologies available to us with this lens, as follows:

SNARK-based. The recent progress in succinct proof systems [BFS20, BCR+19, BBB+18, Gro16] provides a tempting avenue to explore, as a SNARK attesting to the correctness of nonce generation yields a conceptually simple approach. We highlight here that we wish to rely on standard assumptions, the implication being that we would like to use a time-tested and vetted, preferably standardized PRF. While there has been tremendous progress in constructing SNARK/STARK-friendly ciphers [BSGL20], efficiently proving statements involving more traditional non-algebraic ciphers (such as SHA/AES) has remained elusive using any SNARK technology. For instance the fastest such succinct proof system at present (Spartan [Set20]) would require over 100ms to prove a *single* AES computation ($\approx 2^{14}$ R1CS constraints [Kos]) on a modern laptop as per their implementation.

Generic MPC. Advances in generic MPC [KRRW18, HSS17, KPR18] have brought the secure computation of sophisticated cryptographic functions into the realm of practicality. However they are all inherently interactive and randomized (with many being heavily reliant on preprocessing), posing fresh challenges in the deterministic/stateless setting. Additionally even the most advanced constant round techniques [KRRW18, HSS17] require several rounds of interaction, marking a departure from conventional threshold Schnorr which needs only three rounds.

Zero-knowledge for Composite Statements. Chase et al. [CGM16] construct two protocols in the ZKGC paradigm [JKO13] that bind algebraic and non-algebraic bitwise encodings of the same value, so that the algebraic encoding may be used for efficient sigma protocols while the non-algebraic encoding can be used to evaluate a garbled circuit. Roughly, the two methods are as follows, with the following tradeoffs:

1. *Homomorphic bitwise commitments to the witness*: This method produces smaller proofs, and is even extended to the MPC-in-the-head setting by Backes et al. [BHH+19]. However this fundamentally requires exponentiations for each bit of the input, i.e. $O(|q|)$ asymp-

totically and hundreds concretely for our parameter range, which would require many tens of milliseconds at least to compute on commodity hardware. We therefore do not pursue this line further.

2. *Algebraic MAC applied to the witness*: This method produces larger proofs, as the MAC is computed by garbling an $\tilde{O}(s \cdot |q|)$ circuit. However this avoids public key operations (besides OT) and presents a promising direction to investigate.

Equipped with an understanding of the landscape of proof systems, we expand on the results that we summarized in Section 3.3.

5.4 Exponentiation Garbling Gadget

In this section, we give our new garbling gadget that translates a standard Yao-style representation of a binary string (i.e. with wire labels) to an algebraic encoding of the same value in the target elliptic curve group. As we intend to compose this gadget with the Half Gates garbling scheme [ZRE15] we give the construction and proof assuming FreeXOR style keys [KS08]. Consequently we prove security assuming a correlation robust hash function (strictly weaker than circular correlation robustness [CKKZ12] as needed by FreeXOR/HalfGates). Note that this structure is not required by our scheme, and security can easily by proven assuming just PRFs if desired.

Algorithm 5.4.1. $\mathcal{G}_{\mathsf{exp}}$. Privacy-free Exponentiation Garbling Gadget

This scheme allows to garble the gadget $f : \{0,1\}^\eta \mapsto \mathbb{G}$, in particular $f(x) = \langle \mathbf{u}, \mathbf{x} \rangle \cdot G$ where $\mathbf{u} \in (\mathbb{Z}_q^*)^\eta$ is a public vector of group elements, the vector $\mathbf{x}$ is a length η bit string, and $G \in \mathbb{G}$ generates an elliptic curve group $\mathbb{G}$. Note that the garbled output is encoded arithmetically, and as such can not be composed with (i.e. fed as input to) a standard binary circuit garbling scheme. All algorithms make use of the key derivation function KDF.

$\mathsf{Gb}(1^\kappa, g)$**:** .

1. Sample $\Delta \leftarrow \{0,1\}^\kappa$ and $a \leftarrow \mathbb{Z}_q^*$
2. For each $i \in [\eta]$,
   (a) Sample $k_i \leftarrow \{0,1\}^\kappa$
   (b) Compute $b_i = \mathsf{KDF}(i, k_i)$
   (c) Set $\tilde{C}_i = \mathsf{KDF}(i, k_i \oplus \Delta) - (b_i + \mathbf{u}_i \cdot a)$

3. Set $b = \sum_{i\in[\eta]} b_i$ and $B = b \cdot G$

4. Compute encoding information $\mathsf{en} = \left[\Delta, \{k_i\}_{i\in[\eta]}\right]$

5. The decoding information is $\mathsf{de} = (a, B)$

6. Output $\tilde{C}, \mathsf{en}, \mathsf{de}$

$\mathsf{En}(\mathsf{en}, x)$**:** .

1. Parse $\left[\Delta, \{k_i\}_{i\in[\eta]}\right]$ from en, and for each $i \in [\eta]$: set $X_i = k_i \oplus x_i \cdot \Delta$

2. Output $\tilde{X} = \{(x_i, X_i)\}_{i\in[\eta]}$

$\mathsf{Ev}(\tilde{C}, \tilde{X})$**:** .

1. Parse $\{(x_i, X_i)\}_{i\in[\eta]}$ from $\tilde{X}$

2. For each $i \in [\eta]$: Compute $z_i = \mathsf{KDF}(i, X_i) - x_i \cdot \tilde{C}_i$

3. Compute $z = \sum_{i\in[\eta]} z_i$, and output $\tilde{Z} = z \cdot G$

$\mathsf{De}(\mathsf{de}, \tilde{Z})$**:** Parse (a, B) from de and output $a^{-1} \cdot (\tilde{Z} - B)$

We first give the exact definition required of KDF in order to secure the garbling scheme. Informally, KDF is correlation robust if $\mathsf{KDF}(x \oplus \Delta)$ appears random even under adversial choice of x when Δ is chosen uniformly and hidden from the adversary.

Definition 5.4.2 (Correlation Robust Hash Function)**.** *Let the security parameter κ determine a 2κ-bit prime q, and be an implicit parameter in the following stateful oracles $\mathcal{O}_{\mathsf{KDF}}$ and $\mathcal{O}_R$ defined as follows:*

- *$\mathcal{O}_{\mathsf{KDF}}(i, x)$: Upon first invocation, sample $\Delta \leftarrow \{0,1\}^\kappa$. Return $\mathsf{KDF}(i, x \oplus \Delta)$*
- *$\mathcal{O}_R(i, x)$: If not previously queried on x, sample $F(i, x) \leftarrow \mathbb{Z}_q$. Return $F(i, x)$.*

A hash function KDF is correlation robust if $\mathcal{O}_{\mathsf{KDF}}$ and $\mathcal{O}_R$ are computationally indistinguishable to any PPT adversary with unrestricted oracle access.

We are now ready to state the security theorem for $\mathcal{G}_{\mathsf{exp}}$.

Theorem 5.4.3. *Assuming KDF is a correlation robust hash function, $\mathcal{G}_{\mathsf{exp}}$ is a privacy-free garbling scheme for the function $f_{\mathbf{u}}(\mathbf{x}) = \langle \mathbf{u}, \mathbf{x} \rangle \cdot G$.*

Proof. **Correctness.** Observe that for each $i \in [\eta]$ the evaluator computes

$$z_i = \mathsf{KDF}(i, X_i) - x_i \cdot \tilde{C}_i$$

Substituting $\tilde{C}_i = \mathsf{KDF}(i, k_i \oplus \Delta) - (b_i + \mathbf{u}_i \cdot a)$ and $X_i = k_i \oplus x_i \cdot \Delta$ into the above equation, we obtain:

$$z_i = \mathsf{KDF}(i, k_i \oplus x_i \cdot \Delta) - x_i \cdot (\mathsf{KDF}(i, k_i \oplus \Delta) - (b_i + \mathbf{u}_i \cdot a))$$

The above expression therefore simplifies to two cases:

$$z_i = \begin{cases} \mathsf{KDF}(i, k_i) \text{ when } x_i = 0 \\ b_i + \mathbf{u}_i \cdot a \text{ when } x_i = 1 \end{cases}$$

Since $\mathsf{KDF}(i, k_i) = b_i$, we can simplify the above to $z_i = b_i + x_i \cdot \mathbf{u}_i \cdot a$. We therefore have that

$$z = \sum_{i \in [\eta]} z_i = \sum_{i \in [\eta]} (b_i + x_i \cdot \mathbf{u}_i \cdot a) = b + a \cdot \langle \mathbf{u}, \mathbf{x} \rangle$$

And therefore $\tilde{Z} = z \cdot G = B + a \cdot \langle \mathbf{u}, \mathbf{x} \rangle \cdot G$. Clearly the decoding procedure $a^{-1} \cdot (\tilde{Z} - B)$ yields $\langle \mathbf{u}, \mathbf{x} \rangle \cdot G$.

Verifiablity. Revealing en allows each b_i to be computed and $\tilde{C}_i$ to be decrypted, and clearly if every $\tilde{C}_i = \mathsf{KDF}(i, k_i \oplus \Delta) - (b_i + \mathbf{u}_i \cdot a)$ for the same value of a, the values $\tilde{C}, \tilde{X}$ will always evaluate consistently for all inputs.

Authenticity. We prove that the encoded output is unforgeable (i.e. authentic) via hybrid experiments. Recall that the experiment for authenticity of a garbling scheme works as follows: the adversary $\mathcal{A}$ sends a circuit f and input x to the challenger, which then responds with $\tilde{C}, \tilde{X}$ where $\tilde{C}, \mathsf{en}, \mathsf{de} \leftarrow \mathsf{Gb}(f)$ and $\tilde{X} = \mathsf{En}(\mathsf{en}, x)$. If $\mathcal{A}$ is able to produce valid garbled output $\hat{Z}$ such that $\mathsf{De}(\mathsf{de}, \hat{Z}) \neq f(x)$ then the adversary wins.

Hybrid $\mathcal{H}_1$. We first define a hybrid experiment $\mathcal{H}_1$ that changes the way $\tilde{C}, \tilde{X}$ is computed. In particular, $\tilde{C}, \tilde{X}$ are jointly produced using f, x rather than by separate garbling and encoding procedures, as detailed below:

1. Sample $\Delta \leftarrow \{0,1\}^\kappa$ and $a \leftarrow \mathbb{Z}_q^*$
2. For each $i \in [\eta]$,
   (a) Sample $k_i \leftarrow \{0,1\}^\kappa$
   (b) If $\mathbf{x}_i = 0$ then

i. Compute $b_i = \mathsf{KDF}(i, k_i)$

ii. Set $\tilde{C}_i = \mathsf{KDF}(i, k_i \oplus \Delta) - (b_i + \mathbf{u}_i \cdot a)$

(c) Otherwise

i. Compute $b_i = \mathsf{KDF}(i, k_i \oplus \Delta)$

ii. Set $\tilde{C}_i = \mathsf{KDF}(i, k_i) - (b_i + \mathbf{u}_i \cdot a)$

3. Set $b = \sum_{i \in [\eta]} b_i$ and $B = b \cdot G$

4. Compute $\tilde{X} = \{k_i\}_{i \in [\eta]}$

5. The decoding information is $\mathsf{de} = (a, B)$

6. Output $\tilde{C}, \tilde{X}, \mathsf{de}$

The distribution of $\tilde{C}, \tilde{X}$ in this hybrid experiment is identical to the real experiment. Observe that the only change is that the 'active' key (i.e. key seen by the evaluator) on the i^{th} wire is defined to be k_i in $\mathcal{H}_1$, whereas in the real experiment the active key is $k_i \oplus x_i \cdot \Delta$. As the inactive key in both experiments is simply the active key $\oplus\Delta$, this is merely a syntactic change. Therefore we have that for all adversaries $\mathcal{A}$, functions and inputs $f_{\mathbf{u}}, \mathbf{x}$ and any string $\hat{Z}$:

$$\begin{aligned} &\Pr\left[\hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}) : (\tilde{C}, \mathsf{en}, \mathsf{de}) \leftarrow \mathsf{Gb}(f_{\mathbf{u}}), \tilde{X} \leftarrow \mathsf{En}(\mathsf{en}, \mathbf{x})\right] \\ &= \Pr\left[\hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}) : (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_1(f_{\mathbf{u}}, \mathbf{x})\right] \end{aligned} \tag{5.1}$$

Hybrid $\mathcal{H}_2$. In this hybrid experiment, the inactive key is changed from $k_i \oplus \Delta$ to a uniformly random value. In particular, the code for this hybrid experiment is identical to the last except for the following two changes:

	Experiment $\mathcal{H}_1$	Experiment $\mathcal{H}_2$
Step 2(b)ii	$\tilde{C}_i = \mathsf{KDF}(i, k_i \oplus \Delta) - (b_i + \mathbf{u}_i \cdot a)$	$\tilde{C}_i \leftarrow \mathbb{Z}_q$
Step 2(c)i	$b_i = \mathsf{KDF}(i, k_i \oplus \Delta)$	$b_i \leftarrow \mathbb{Z}_q$

A distinguisher for the values $(\tilde{C}, \tilde{X})$ produced by $\mathcal{H}_1$ and $\mathcal{H}_2$ immediately yields a distinguisher for the correlation robustness property of KDF. The reduction simply runs the code of $\mathcal{H}_1$, and in place of using KDF in Step 2(b)ii and Step 2(c)i, it queries the challenge oracle $\mathcal{O}$ with the same arguments. In the case that $\mathcal{O} = \mathcal{O}_{\mathsf{KDF}} = \mathsf{KDF}$ this exactly produces the distribution per $\mathcal{H}_1$, and in the case $\mathcal{O} = \mathcal{O}_R$ (i.e. truly random function) the distribution per $\mathcal{H}_2$ is exactly produced, resulting in a lossless reduction to the correlation robustness

property of KDF. We therefore have that there is a negligible function negl such that for all PPT adversaries $\mathcal{A}$ and $\hat{Z} \in \mathbb{G}$:

$$\left| \begin{array}{l} \Pr[\hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}) : (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_2(f_{\mathbf{u}}, \mathbf{x})] \\ -\Pr[\hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}) : (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_1(f_{\mathbf{u}}, \mathbf{x})] \end{array} \right| \leq \mathsf{negl}(\kappa) \tag{5.2}$$

Hybrid $\mathcal{H}_3$. This hybrid experiment is the same as the last, with the exception that $\tilde{C}_i \leftarrow \mathbb{Z}_q$ for each $i \in [\eta]$. This differs from Step 2(c)ii, which computes $\tilde{C}_i = \mathsf{KDF}(i, k_i) - (b_i + \mathbf{u}_i \cdot a)$ when $\mathbf{x}_i = 1$. However in $\mathcal{H}_2$ when $\mathbf{x}_i = 1$ the value b_i is sampled uniformly from $\mathbb{Z}_q$ and never exposed anywhere else in $\tilde{C}, \tilde{X}$ anyway, effectively acting as a one-time pad. Therefore the distribution of $\tilde{C}, \tilde{X}$ remains unchanged from $\mathcal{H}_2$. In particular,

$$\begin{array}{c} \Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = Y : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_2(f_{\mathbf{u}}, \mathbf{x})\right] \\ = \Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = Y : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_3(f_{\mathbf{u}}, \mathbf{x})\right] \end{array} \tag{5.3}$$

Hybrid $\mathcal{H}_4$. This experiment is the same as the last, except that the definition of the decoding information $\mathsf{de} = (a, B)$ is postponed to after $\tilde{C}, \tilde{X}$ are defined. This induces no change in the distribution of $\tilde{C}, \tilde{X}$ as in $\mathcal{H}_3$ they are computed independently of a, B. The value a is derived the same way (uniformly sampled from $\mathbb{Z}_q^*$), whereas now B is computed as $B = Z - a \cdot Y$ where $Y = f_{\mathbf{u}}(\mathbf{x})$ and $Z = \mathsf{Ev}(\tilde{C}, \tilde{X})$. The distribution of (a, B) is unchanged from $\mathcal{H}_3$, note that by definition $\mathsf{De}(\mathsf{de}, \mathsf{Ev}(\tilde{C}, \tilde{X})) = f_{\mathbf{u}}(\mathbf{x})$ in both experiments. Therefore:

$$\begin{array}{c} \Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = Y : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_3(f_{\mathbf{u}}, \mathbf{x})\right] \\ = \Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = Y : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_4(f_{\mathbf{u}}, \mathbf{x})\right] \end{array} \tag{5.4}$$

We can now bound the probability that an adversary is able to forge an output: consider any $\hat{Y} \in \mathbb{G}$ such that $\hat{Y} \neq Y$. In order to induce $\mathsf{De}(\mathsf{de}, \hat{Z}) = \hat{Y}$, the adversary $\mathcal{A}(\tilde{C}, \tilde{X})$ must output $\hat{Z}$ such that $\hat{Z} - Z = a(\hat{Y} - Y)$. As $\hat{Y} - Y \neq 0$ and a is sampled uniformly from $\mathbb{Z}_q^*$ only after $\hat{Z}, Z, \hat{Y}, Y$ have already been defined, the probability that this relation is satisfied is exactly $1/(q-1)$.

More precisely, for any $f_{\mathbf{u}}$, $\mathbf{x} \in \{0,1\}^\eta$, $\hat{Y} \in \mathbb{G}$ such that $\hat{Y} \neq f_{\mathbf{u}}(\mathbf{x})$ and unbounded adversary $\mathcal{A}$,

$$\Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = \hat{Y} : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_4(f_{\mathbf{u}}, \mathbf{x})\right] = 1/(q-1) \tag{5.5}$$

For our choice of parameters, we have $1/(q-1) \leq 2^{-\kappa}$ which is negligible in κ.

Combining equations 5.1-5.5 we conclude that for any $f_{\mathbf{u}}$, $\mathbf{x} \in \{0,1\}^\eta$, $\hat{Y} \in \mathbb{G}$ such that $\hat{Y} \neq f_{\mathbf{u}}(\mathbf{x})$ and PPT adversary $\mathcal{A}$, the following probability is negligible in κ:

$$\Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = \hat{Y} : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}), (\tilde{C}, \mathsf{en}, \mathsf{de}) \leftarrow \mathsf{Gb}(f_{\mathbf{u}}), \tilde{X} \leftarrow \mathsf{En}(\mathsf{en}, \mathbf{x})\right]$$

The garbling scheme $\mathcal{G}_{\mathsf{exp}}$ is therefore correct, verifiable, and authentic, and so the theorem is hence proven.

□

5.5 Committed OT from UC Commitments

In this section, we give the details of our approach to constructing our committed OT from UC commitments. Recall that we need an OT protocol where the receiver commits to its choice bits during an offline phase, and the sender is able to send (and subsequently open) message pairs relative to the same choice bit. This is because the receiver's choice bits will correspond to the prover's witness (i.e. the PRF key for nonce derivation) which can be committed once during key generation; signing corresponds to proving different statements about the same witness.

We give here the exact functionality $\mathcal{F}^*_{\mathsf{COT}}$ for unlockable oblivious transfer.

Functionality 5.5.1. $\mathcal{F}^*_{\mathsf{COT}}$. Unlockable Committed OT

This functionality allows a receiver R to commit to a choice bit, and subsequently allows a sender S to send indexed message pairs, of which the receiver obtains the one corresponding its choice bit. S provides a key key to lock its messages, which R may present to unlock both messages. The sender's messages remain secure iff an index is never reused for two different message pairs. Additionally any index that is 'revealed' subsequently offers no security when reused. All messages are adversarially delayed.

Choose: Upon receiving $(sid, \texttt{choose}, b)$ from R, and if $b \in \{0,1\}$ and no such message was previously received, store $(sid, \texttt{chosen}, b)$ in memory and send $(\texttt{chosen})$ to S.

Transfer: Upon receiving $(sid, \texttt{transfer}, \mathsf{ind}, \mathsf{key}, m_0, m_1)$ from S, if $(sid, \texttt{chosen}, b)$ exists in memory then:

- If R is not corrupt, store $(sid, \mathsf{ind}, \mathsf{key}, m_0, m_1)$ and send $(sid, \texttt{message}, \mathsf{ind}, m_b)$ to R
- Otherwise:
  1. If $(sid, \mathsf{ind}, m'_0, m'_1)$ exists in memory such that $m_0 \neq m'_0$ or $m_1 \neq m'_1$ then send $(sid, \texttt{reused-index}, m_0, m_1, m'_0, m'_1)$ to R
  2. If $(sid, \texttt{index-used}, \mathsf{ind})$ exists in memory, then send $(sid, \texttt{revealed- index}, \mathsf{ind}, m_0, m_1)$ to R.

3. If neither of the previous conditions hold, then store $(sid, \mathsf{ind}, \mathsf{key}, m_0, m_1)$ and send $(sid, \mathtt{message}, \mathsf{ind}, m_b)$ to R.

Reveal: Upon receiving $(sid, \mathtt{reveal}, \mathsf{ind}, \mathsf{key})$ from R, if $(sid, \mathsf{ind}, \mathsf{key}, m_0, m_1)$ exists in memory, then send $(sid, \mathtt{messages}, m_0, m_1)$ to R. Store $(sid, \mathtt{index\text{-}used}, \mathsf{ind})$ in memory.

Why is this challenging? Consider the following simple attempt at instantiating this object: during the preprocessing phase, the sender samples two PRF keys k_0, k_1, of which the receiver obtains k_b via OT. In order to transmit a message pair m_0, m_1 online, assuming some public instance-specific information x, the sender computes $c_0 = \mathsf{F}_{k_0}(x) \oplus m_0,\ c_1 = \mathsf{F}_{k_1}(x) \oplus m_1$ and sends them to the receiver, who is able to decrypt m_b. In order to 'open' the messages, the sender gives $\mathsf{F}_{k_0}(x), \mathsf{F}_{k_1}(x)$ to the receiver, who then obtains m_{1-b}. While this protects the sender against a malicious receiver, the flaw lies in that it doesn't bind the sender to any particular message pair m_0, m_1. For instance during the opening phase, the sender could provide $\mathsf{F}_{k_0}(x), r^*$ (for some $r^* \neq \mathsf{F}_{k_1}(x)$). If the receiver's choice bit was 0, it does not notice this deviation and outputs $m_1^* = c_1 \oplus r^*$, as opposed to $m_1 = c_1 \oplus \mathsf{F}_{k_1}(x)$ which would have been the output if the receiver's choice bit was 1. Inconsistencies of this flavour propagate upwards to induce selective failure attacks in the ZKGC protocol. We leave the exact attack implicit. This issue is easily solved by using a PRF which allows outputs to be efficiently verified such as a Verifiable Random Function [MRV99]. However to the best of our knowledge, all such known primitives require public key operations, which would defeat the purpose of having moved the OTs offline.

To recap our idea: assume that $\mathcal{C}$ is a commitment scheme that permits *straight-line extraction*. In particular there exists an extractor which, given a commitment and an extraction key ek (corresponding to the commitment key ck), outputs the committed message. This is a property that is conducive to arguing security under concurrent composition [Can01]. However in the 'real' protocol no party has ek; the receiver has a verification key vk which it uses to validate openings to commitments, but the existence of ek is only required for the proof. We will characterize the commitment scheme as a collection of concrete algorithms (rather than working in an $\mathcal{F}_{\mathsf{Commit}}$ hybrid model) and so in principle the trapdoor ek can created by a generic setup functionality and given to the receiver. We use such a commitment scheme to realize the notion of committed OT that we need as follows: create two pairs of keys $(\mathsf{ck}_0, \mathsf{vk}_0, \mathsf{ek}_0)$ and $(\mathsf{ck}_1, \mathsf{vk}_1, \mathsf{ek}_1)$, and provide sender S with both $\mathsf{ck}_0, \mathsf{ck}_1$ and receiver R with $\mathsf{ek}_b, \mathsf{vk}_{1-b}$. In order to send a message pair m_0, m_1, S commits to m_0 using ck_0 and m_1 using ck_1. Then R is able to extract m_b using ek_b immediately. Subsequently when it's

time to reveal both messages, S provides decommitment information for m_0, m_1, and R uses vk_{1-b} to validate m_{1-b}.

In more detail, the commitment scheme $\mathcal{C}$ comprises the following algorithms:

- One-time setup:
  - $\mathsf{Gen\text{-}ck}(1^\kappa; \rho^S) \mapsto \mathsf{ck}$. Samples the committer's key with randomness ρ^S.
  - $\mathsf{Gen\text{-}vk}(\mathsf{ck}; \rho^R) \mapsto \mathsf{vk}$. Samples the receiver's verification key using ck with randomness ρ^R.
  - $\mathsf{Gen\text{-}ek}(\mathsf{ck}) \mapsto \mathsf{ek}$. Determines the extraction key given ck.
  - $\mathsf{Gen\text{-}td}(\mathsf{vk}) \mapsto \mathsf{td}$. Determines the trapdoor for equivocation given vk.
- Per message with index ind:
  - $\mathsf{Commit}(\mathsf{ck}, \mathsf{ind}, m) \mapsto \mathsf{C}, \delta$. Produces commitment C and decommitment information δ for message m and index ind.
  - $\mathsf{DecomVrfy}(\mathsf{vk}, \mathsf{ind}, \mathsf{C}, \delta, m) \mapsto \{0, 1\}$. Commitment verification by R.
  - $\mathsf{Ext}(\mathsf{ek}, \mathsf{ind}, \mathsf{C}) \mapsto m \cup \{\perp\}$. Extracts the committed message from C.
  - $\mathcal{S}_{\mathsf{Com},\mathsf{R}^*}(\mathsf{td})$. A simulator that produces and equivocates commitments.

Rather than enumerating a series of definitions that the scheme must satisfy, we use the above interface to construct a protocol, and require that the protocol must UC-realize our commitment functionality. The structure of the commitment functionality $\mathcal{F}_{\mathsf{Com}}$ and the protocol π_{Com} and Simulator $\mathcal{S}_{\mathsf{Com}}$ are straightforward in their usage of $\mathcal{C}$. Protocol π_{Com} makes use of a helper functionality $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$ which simply runs the one-time setup algorithms. We give the formal details in Appendix B.1.

Commitment schemes that are of interest to us allow protocol π_{Com} to be simulated by simulator $\mathcal{S}_{\mathsf{Com}}$ with respect to functionality $\mathcal{F}_{\mathsf{Com}}$. Also note that by virtue of the definition, commitment is inherently stateless; no state has to be maintained across commitment instances that use different ind values.

Definition 5.5.2. *A commitment scheme $\mathcal{C}$ is a **preprocessable UC commitment** if protocol $\pi_{\mathsf{Com}}[\mathcal{C}]$ can be simulated by $\mathcal{S}_{\mathsf{Com}}[\mathcal{C}]$ with respect to functionality $\mathcal{F}_{\mathsf{Com}}$ in the UC experiment where an adversary statically corrupts up to one party, in the $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$-hybrid model.*

We stress that while we refer to $\mathcal{C}$ as the preprocessable UC commitment scheme, the actual protocol for the UC experiment is $\pi_{\mathsf{Com}}[\mathcal{C}]$, which is merely a wrapper for the algorithms specified by $\mathcal{C}$.

Instantiating $\mathcal{F}_{\mathsf{Com}}$

Efficiently instantiating the UC commitment functionality (of which $\mathcal{F}_{\mathsf{Com}}$ is a relaxation) has been studied extensively in the literature [DN02, Lin11, CJS14]. However the subset of such works most relevant here are those that operate in the offline-online paradigm, where expensive message-independent public key operations are pushed to an offline phase and (de)committments online only require cheap symmetric key operations. Such protocols have been constructed in a line of works [DDGN14, GIKW14, CDD+15, FJNT16] where a number of oblivious transfers are performed offline to establish correlations, and (de)committing online derives security from the fact that the receiver knows some subset (but not all) of the sender's secrets. Some of these works [CDD+15, FJNT16] are quite practical; their technique is roughly to have the sender commit to a message by first encoding it using an error correcting code, then producing additive shares of each component of the resulting codeword, and finally sending the receiver each additive share encrypted by a pseudorandom one-time pad derived by extending a corresponding PRG seed. The receiver has some subset of these seeds (chosen via OT offline) and obtains the corresponding shares of the codeword. The committed message stays hidden as the receiver is missing one additive share of each component. To decommit, the sender reveals the entire codeword and its shares, and the receiver checks consistency with the shares it already knows. Soundness comes from the property that changing the committed message requires changing so many components of the codeword that the receiver will detect such a change with overwhelming probability. The trapdoor for extraction is the entire set of PRG seeds that are used to encrypt the codeword components. As the sender must encrypt a value that is close to a codeword using these seeds, the extractor is able to decrypt and decode the near-codeword to retrieve the committed message. Extraction is possible as the simulator knows all PRG seeds, and the sender must have encrypted a value sufficiently close to a real codeword in order to have a non-negligible chance of the receiver accepting it later.

Cascudo et al. [CDD+15] report a concretely efficient instantiation of this idea by using binary BCH codes. However existing constructions are designed to amortize the cost of (de)committing large numbers of messages, and as such they are heavily reliant on maintaining state for the PRG. It is feasible to modify their constructions to be stateless by the standard method of replacing the PRG with a PRF, but the resulting cost per instance would save little compared to exponentiation; for instance the protocol of Frederiksen et al. [FJNT16] would require over 800 PRF invocations per instance at a 40 bit security level. While this cost disappears over many simultaneous instances in their setting, we unfortunately can not amortize our costs as independent instances must not share state.

Our Technique. Our commitment scheme essentially implements the same high-level idea, but with a repetition code. The sender S has ℓ PRF keys $k_1, \cdots, k_\ell$, of which the receiver R is given a random subset of $\ell - 1$ (say all but $i \in [\ell]$). In order to commit to a message μ for index ind, S sends $\mathsf{F}_{k_1}(\mathsf{ind}) \oplus \cdots \oplus \mathsf{F}_{k_\ell}(\mathsf{ind}) \oplus \mu$ to the receiver. In order to decommit, S reveals μ and $\mathsf{F}_{k_1}(\mathsf{ind}), \cdots, \mathsf{F}_{k_\ell}(\mathsf{ind})$, given which R computes $\mathsf{F}^*_{k_i} = \mu \bigoplus_{j \in [\ell] \setminus i} \mathsf{F}_{k_j}(\mathsf{ind})$ and verifies that it matches F_{k_i} claimed by S. Intuitively, S has to guess exactly which key R is missing in order to fool it. This has soundness error $1/\ell$, however simply repeating this procedure sufficiently many times in parallel (with independent keys) boosts the protocol to have negligble soundness error. This description omits some details, such as how the repetitions are bound together, and optimizing communication, so we describe the commitment scheme itself in terms of the language we laid out earlier.

Algorithm 5.5.3. $\mathcal{C}$. Commitment scheme

This set of algorithms instantiates a commitment scheme $\mathcal{C}$. The security parameter κ fixes statistical security parameter s and integers ℓ and r such that $\mathsf{r}\log_2(\ell) = s$. The (de)commitment protocols make use of a random oracle RO for equivocation, but notably the extractor does not observe queries to the RO (meaning that it can be run without a backdoor for RO). The protocols additionally use a collision resistant hash function CRHF.

$\mathsf{Gen\text{-}ck}(1^\kappa; \rho^S)$**:** .

1. For each $j \in [\mathsf{r}]$ and $l \in [\ell]$, sample $k_{j,l} \leftarrow \{0,1\}^\kappa$
2. Sample $k^* \leftarrow \{0,1\}^\kappa$
3. Output $\mathsf{ck} = k^*, \{k_{j,l}\}_{j \in [\mathsf{r}], l \in [\ell]}$

$\mathsf{Gen\text{-}vk}(\mathsf{ck}; \rho^R)$**:** .

1. Parse $\{k_{j,l}\}_{j \in [\mathsf{r}], l \in [\ell]}$ from ck, and for each $j \in [\mathsf{r}]$, sample integer $i_j \leftarrow [\ell]$
2. Output $\mathsf{vk} = \left\{(k_{j,l})_{l \in [\ell] \setminus i_j}\right\}_{j \in [\mathsf{r}]}$

$\mathsf{Gen\text{-}ek}(\mathsf{ck})$**:** Output ck
$\mathsf{Gen\text{-}td}(\mathsf{vk})$**:** Output $\{i_j\}_{j \in [\ell]}$
$\mathsf{Commit}^{\mathsf{RO}}(\mathsf{ck}, \mathsf{ind}, m)$**:** .

1. Compute $\mu = \mathsf{F}_{k^*}(\mathsf{ind})$, and for each $j \in [\mathsf{r}]$ set $\mathsf{ct}_j = \mu \bigoplus_{l \in [\ell]} \mathsf{F}_{k_{j,l}}(\mathsf{ind})$

2. Set $\mathsf{ct} = \{\mathsf{ct}_j\}_{j\in[\mathsf{r}]}$, $\mathsf{h} = \mathsf{RO}(\mu)$, $\xi = \mu \oplus m$

3. Set $\delta = \mathsf{CRHF}\left(\left\{\mathsf{F}_{k_{j,l}}(\mathsf{ind})\right\}_{j\in[\mathsf{r}],l\in[\ell]}\right)$, and output $\mathsf{C} = (\mathsf{ct}, \mathsf{h}, \xi), \delta$

$\mathsf{DecomVrfy}(\mathsf{vk}, \mathsf{ind}, \mathsf{C}, \delta, m)$**:** .

1. Parse $\{i_j\}_{j\in[\ell]} = \mathsf{vk}$, and $\mathsf{ct}, \mathsf{h}, \xi$ from C

2. Compute $\mu = m \oplus \xi$ and verify $\mathsf{RO}(\mu) \stackrel{?}{=} \mathsf{h}$

3. For each $j \in [\mathsf{r}]$ compute $\mathsf{F}^*_{j,k_{i_j}} = \mu \oplus \mathsf{ct}_j \bigoplus_{l\in[\ell]\setminus i_j} \mathsf{F}_{k_{j,l}}(\mathsf{ind})$

4. For each $j \in [\mathsf{r}]$, set $\mathsf{F}[j,l] = \mathsf{F}_{k_{j,l}}(\mathsf{ind})$ for $l \in [\ell]\setminus i_j$ and $\mathsf{F}[j,i_j] = \mu \oplus \mathsf{ct}_j \bigoplus_{l\in[\ell]\setminus i_j} \mathsf{F}_{k_{j,l}}(\mathsf{ind})$

5. Verify $\delta \stackrel{?}{=} \mathsf{CRHF}\left(\{\mathsf{F}[j,l]\}_{j\in[\mathsf{r}],l\in[\ell]}\right)$

$\mathsf{Ext}(\mathsf{ek}, \mathsf{ind}, \mathsf{C})$**:** .

1. Parse $\{k_{j,l}\}_{j\in[\mathsf{r}],l\in[\ell]}$ from ck, and $\mathsf{ct}, \mathsf{h}, \xi$ from C

2. For each $j \in [\mathsf{r}]$, compute $\mu^*_j = \mathsf{ct}_j \bigoplus_{l\in[\ell]} \mathsf{F}_{k_{j,l}}(\mathsf{ind})$

3. If $\exists j \in [\mathsf{r}]$ such that $\mathsf{RO}(\mu^*_j) = \mathsf{h}$, then output $m = \mu^*_j \oplus \xi$

4. If no such μ^*_j exists, then output $\perp$

$\mathcal{S}_{\mathsf{Com},\mathsf{R}^*}(\mathsf{td})$**:** See proof of Theorem 5.5.4.

Theorem 5.5.4. *Assuming* F *is a pseudorandom function and* CRHF *is a collision resistant hash function,* $\mathcal{C}$ *is a preprocessable UC commitment in the local random oracle model.*

Proof. (Sketch.) Recall that the actual protocol for the UC experiment is $\pi_{\mathsf{Com}}[\mathcal{C}]$. We first argue why the extractor Ext succeeds except with probability 2^{-s}. First note that except with negligible probability, there is at most one μ queried to RO such that $\mathsf{RO}(\mu) = \mathsf{h}$. The extractor iteratively computes $\mu^*_j = \mathsf{ct}_j \bigoplus_{l\in[\ell]} \mathsf{F}_{k_{j,l}}(\mathsf{ind})$ to find this μ. We analyze the exact event in which the extractor fails but the sender produces an accepting decommitment m^*, δ^*. Define $\mu^* = m \oplus \xi$. Consider the state induced by running $\mathsf{DecomVrfy}$ on these inputs but a pair of distinct $\mathsf{vk}, \mathsf{vk}'$: as $\mathsf{vk} \neq \mathsf{vk}'$ there must exist $j \in [\ell]$ such that $i_j \neq i'_j$ (where i, i' are parsed from $\mathsf{vk}, \mathsf{vk}'$ respectively). As the extractor failed, we know that

$$\mathsf{ct}_j \bigoplus_{l\in[\ell]} \mathsf{F}_{k_{j,l}}(\mathsf{ind}) \neq \mu^*$$

This means that when using vk, $\mathsf{DecomVrfy}$ computes

$$\mathsf{F}[j, i_j] = \mu^* \oplus \mathsf{ct}_j \bigoplus_{l \in [\ell] \setminus i_j} \mathsf{F}_{k_{j,l}}(\mathsf{ind}) \neq \mathsf{F}_{k_{j,i_j}}(\mathsf{ind})$$

whereas when using vk', $\mathsf{DecomVrfy}$ computes $\mathsf{F}[j, i_j] = \mathsf{F}_{k_{j,i_j}}(\mathsf{ind})$. As this induces a disagreement about the set $\{\mathsf{F}[j, l]\}_{j \in [\mathsf{r}], l \in [\ell]}$ when using $\mathsf{vk}, \mathsf{vk}'$, $\mathsf{DecomVrfy}$ will not accept on both inputs (unless there's a collision in CRHF). As there are 2^s different choices of vk and no two choices lead to $\mathsf{DecomVrfy}$ accepting the same m^*, δ^*, the probability that the adversary is successful in inducing the extractor to fail is at most 2^{-s}.

Equivocation is easier to show: the simulator can run the honest algorithm with a dummy message, and by PRF security the value μ is hidden from the verifier (as vk omits one PRF key in each set). In order to equivocate to a message m, the simulator simply programs RO on input μ so that $\mathsf{RO}(\mu) \oplus \xi = m$. □

How to implement the setup? Observe that the structure of the verification key is to choose all but one out of the ℓ keys in each of the r batches. This is directly achieved by r invocations of $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{OT}}$.

Efficiency. A commitment to a message m (assume $|m| = \kappa$) is of size $(\mathsf{r} + 3) \cdot \kappa$ bits, and in terms of computation requires $\mathsf{r} \cdot \ell$ PRF evaluations and hashing a $\mathsf{r} \cdot \ell \cdot \kappa$ bit message via CRHF. Decommitment requires the same effort.

Parameters. Looking ahead, we will introduce a privacy amplifying optimization in the ZKGC protocol so that for s bits of statistical security, the receiver's security in the Committed OT protocol it uses (and therefore soundness of the Commitment scheme under the hood) need only achieve $s/2$ bits of statistical security. We therefore calibrate our parameters here appropriately. A reasonable instantiation of parameters would be $\ell = 4$, $s = 30$, $\kappa = 128$, and $\mathsf{r} = 15$ (i.e. a 30-bit statistical soundness level) with AES-128 as the PRF, and SHA-512 as the CRHF and RO. This means that a single commitment to a 128-bit message requires 288 bytes (32 bytes to decommit), 60 AES-128 evaluations, and hashing a 0.96 kilobyte message via SHA-512. The work done by R in verifying a commitment is almost the same. Looking ahead, we will use a pair of these commitments to replace a single OT instance, providing a significant improvement in computation time.

5.5.1 Committed OT from Preprocessable UC Commitments

Using commitment scheme $\mathcal{C}$, we now have an clean template for a protocol to build committed OT. We first define a helper functionality $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$ to handle the preprocessing stage. Intuitively, $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$ samples two commitment keys $\mathsf{ck}_0, \mathsf{ck}_1$ for the sender and corresponding

verification and extraction keys $\mathsf{vk}_0, \mathsf{vk}_1, \mathsf{ek}_0, \mathsf{ek}_1$, and gives $\mathsf{vk}_0, \mathsf{vk}_1, \mathsf{ek}_b$ to the receiver upon its choice of bit b. The formalism is straightforward and so we postpone it to Appendix B.2. Unfortunately it is unclear how to generically construct $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$ using the commitment scheme, but for our specific case we can construct a custom protocol based on the same Bellare-Micali construction that we used for $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{OT}}$. We give the exact construction in Appendix B.2.

Protocol 5.5.5. $\pi_{\mathsf{COT}}[\mathcal{C}]$**. Committed Oblivious Transfer**

This protocol is run between a sender S and a receiver R, and is parameterized by a commitment scheme $\mathcal{C}$. This protocol makes use of the ideal oracle $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$ and random oracle $\mathsf{RO} : \{0,1\}^* \mapsto \{0,1\}^{4\kappa}$.

Setup: R has private input $b \in \{0,1\}$

1. S and R send $(sid, \texttt{init})$ to $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$
2. R additionally sends $(\texttt{choose}, b)$ to $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$, and receives $(sid, \texttt{keys}, \mathsf{ek}_b, \mathsf{vk}_0, \mathsf{vk}_1)$ in response.
3. S receives $(sid, \texttt{ck-keys}, \mathsf{ck}_0, \mathsf{ck}_1)$ from $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$

Transfer: S has private inputs m_0, m_1, key, and ind is public input.

1. S computes $\mathsf{C}_0, \delta_0 = \mathsf{Commit}(\mathsf{ck}_0, \mathsf{ind}, m_0)$ and $\mathsf{C}_1, \delta_1 = \mathsf{Commit}(\mathsf{ck}_1, \mathsf{ind}, m_1)$
2. S encrypts the decommitment information with key as $\nu = \mathsf{RO}(\mathsf{key}) \oplus (m_0, \delta_0, m_1, \delta_1)$
3. S sends $\mathsf{C}_0, \mathsf{C}_1, \nu$ to R
4. R outputs $m_b = \mathsf{Ext}(\mathsf{ek}_b, \mathsf{ind}, \mathsf{C}_b)$

Reveal: R does the following with inputs ind and key:

1. R computes $(m_0, \delta_0, m_1, \delta_1) = \mathsf{RO}(\mathsf{key}) \oplus \nu$
2. R outputs $\mathsf{DecomVrfy}(\mathsf{vk}_0, \mathsf{ind}, \mathsf{C}_0, \delta_0, m_0) \wedge \mathsf{DecomVrfy}(\mathsf{vk}_1, \mathsf{ind}, \mathsf{C}_1, \delta_1, m_1)$

Theorem 5.5.6. *Assuming $\mathcal{C}$ is a preprocessable UC commitment (Def. 5.5.2), protocol π_{COT} UC-realizes $\mathcal{F}_{\mathsf{COT}}^{*}$ in the presence of an adversary corrupting up to one party, in the $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$-hybrid random oracle model.*

The theorem directly follows from the definition of preprocessable UC commitments, and the fact that encryptions with the random oracle carry no information until the correct pre-image is queried.

Efficiency

There are three components to analyze: the setup, transfer, and reveal phases.

Setup. We do not analyze the exact cost of setup, beyond that it requires $O(\ell \mathsf{r} \kappa / \log \kappa)$ curve multiplications, and as many field elements transmitted.

Transfer and Reveal. This is the important metric, as the transfer and reveal phases are executed when a message has to be signed. A transfer consists of two independent instances of preprocessable UC commitments for S, of which R simply receives one and runs Ext on the other. A reveal requires no work for S, and two decommitment verifications for R. In our specific instantiation, the work done by S when committing and R when verifying is roughly the same. Additionally R can reuse the work of Ext in verifying a commitment. Based on Section 5.5, the work done by each party in total for a transfer and reveal of a message pair is 120 AES invocations, and hashing a 1.92KB message via SHA-512. The bandwidth consumed is two UC commitments and their decommitments, so 0.64KB. Note that these parameters are for a 30-bit statistical security level, which is inadequate by itself, but will be sufficient in the ZKGC context due to a privacy amplifying technique.

5.6 Provable Nonce Derivation

In order to clarify the target, we give the ideal functionality $\mathcal{F}_{\mathsf{F},\mathbb{G}}$ for proving deterministic nonce derivation, with a conditional disclosure property woven in.

Functionality 5.6.1. $\mathcal{F}_{\mathsf{F},\mathbb{G}}$. Deterministic Nonce Derivation

This functionality is accessed by a prover P and a verifier V, and is parameterized by the keyed function $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^\kappa \mapsto \mathbb{Z}_q$, and the group $(\mathbb{G}, G, q)$. In principle, the public instance $x = (m, R^*_m)$ for which the prover has witness $w = k$ satisfies relation $f(x, w)$ only when $R^*_m = \mathsf{F}_k(m) \cdot G$. All messages are adversarially delayed.

Key Generation: This phase is run exactly once for each *sid*. Any requests to the functionality with an *sid* for which Key Generation has not yet been run are ignored.

1. Wait to receive $(sid, \texttt{input-key}, k)$ from P.
2. If $k \in \{0,1\}^\kappa$, then store $(sid, \texttt{key}, k)$ and send $(sid, \texttt{initialized})$ to V.

Verify Nonce: Upon receiving $(sid, \texttt{verify-nonce}, m, R_m^*)$ from P and $(sid, \texttt{verify-nonce}, m, R_m^*, z)$ from V, if $(sid, \texttt{key}, k)$ exists in memory, $m \in \{0,1\}^\kappa$, and $R_m^* \in \mathbb{G}$ then:

1. Compute $r_m = \mathsf{F}_k(m)$ and $R_m = r_m \cdot G$.
2. If $R_m^* \stackrel{?}{=} R_m$ then send $(sid, \texttt{secret}, m, z)$ to P and then $(sid, \texttt{verified}, m, R_m^*)$ to V. Otherwise send $(sid, \texttt{fail}, m, R_m^*)$ to V.

This is essentially a specific instantiation of the standard zero-knowledge functionality, with the exception that the prover commits its witness w first, and subsequently multiple statements x are supplied to the functionality, which verifies that $R(x, w) = 1$. This is directly achieved by replacing the committed OT functionality used by ZKGC with $\mathcal{F}_{\mathsf{COT}}^*$ which allows the receiver to commit to a choice bit and subsequently receive/open multiple message pairs independently without ever revealing the choice bit. Note that the circuit to be garbled ($C(k, x) = \mathsf{F}_k(x) \cdot G$) is supported by HalfGates with our garbling gadget. Finally, the disclosure of the secret z conditioned on the validity of the statement/witness is the same as the technique introduced by Ganesh et al. [GKPS18]. We give the explicit protocol below.

Protocol 5.6.2. $\pi_{\mathsf{F}\cdot\mathbb{G}}$. **Deterministic Nonce Derivation**

This protocol is parameterized by the security parameter κ, elliptic curve group $(\mathbb{G}, G, q)$ ($|q| = 2\kappa$), PRF $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^\kappa \mapsto \mathbb{Z}_q$, vector $\mathbf{u}$, and two parties Prover P and Verifier V. This protocol makes use of the ideal oracle $\mathcal{F}_{\mathsf{COT}}^*$, random oracle $\mathsf{RO} : \{0,1\}^* \mapsto \mathbb{Z}_q$, and garbling scheme $\mathcal{G}$. Denote by $\mathsf{F} \cdot \mathbb{G}$ the circuit $C(k, x)$ that computes bit vector $\mathbf{y} = \mathsf{F}_k(x)$ via the Boolean representation of F, and outputs $Y = \langle \mathbf{u}, \mathbf{y} \rangle \cdot G$

Key Registration: Run once, with P using private input k:

1. P computes the bit decomposition of k as $k_0 k_1 k_2, \cdots k_\kappa$
2. For each $i \in [\kappa]$, P sends $(i, \texttt{choose}, k_i)$ to $\mathcal{F}_{\mathsf{COT}}^*$
3. V waits to receive $(i, \texttt{chosen})$ from each $i \in [\kappa]$
4. V samples $k_V \leftarrow \{0,1\}^\kappa$

Derive Nonce: With common input $m \in \{0,1\}^\kappa$ and $R_m \in \mathbb{G}$, and secret input z for V:

1. P and V compute $\mathsf{ind} = \mathsf{CRHF}(m, R_m)$

2. V does the following:

   (a) Derive randomness needed for garbling, $\rho_{\mathcal{G}} = \mathsf{RO}(k_V, \mathsf{ind}, \mathtt{Gb})$

   (b) Generate garbled circuit $\tilde{C}, \mathsf{en}, \mathsf{de} = \mathsf{Gb}(\mathsf{F} \cdot \mathbb{G}; \rho_{\mathcal{G}})$

   (c) Parse encoding and decoding information $(\mathsf{en}_{i,0}, \mathsf{en}_{i,1})_{i\in[\kappa]} = \mathsf{en}$ and $(a, B) = \mathsf{de}$ respectively

   (d) Compute the OT key $\mathsf{key} = a \cdot R_m + B$

   (e) Compute the CDS ciphertext $\zeta = \mathsf{RO}(\mathsf{key}) \oplus z$

   (f) Send $(\tilde{C}, \zeta)$ to P

   (g) For each $i \in [\kappa]$, send $(i, \mathtt{transfer}, \mathsf{ind}, \mathsf{key}, \mathsf{en}_{i,0}, \mathsf{en}_{i,1})$ to $\mathcal{F}^*_{\mathsf{COT}}$

3. Upon receiving $(\tilde{C}, \zeta)$ from V and $(i, \mathtt{message}, \mathsf{ind}, \mathsf{en}_{i,k_i})$ for each $i \in [\kappa]$ from $\mathcal{F}^*_{\mathsf{COT}}$, P does the following:

   (a) Assemble garbled input $\tilde{X} = (\mathsf{en}_{i,k_i})_{i\in[\kappa]}$ and evaluate the garbled circuit to obtain $\tilde{Y} = \mathsf{Ev}(\tilde{C}, \tilde{X})$. Set $\mathsf{key} = \tilde{Y}$

   (b) For each $i \in [\kappa]$: Send $(i, \mathsf{ind}, \mathsf{key})$ to $\mathcal{F}^*_{\mathsf{COT}}$ and receive $(\mathsf{en}_{i,0}, \mathsf{en}_{i,1})$ in response

   (c) Assemble encoding information $\mathsf{en} = (\mathsf{en}_{i,0}, \mathsf{en}_{i,1})_{i\in[\kappa]}$

   (d) Verify that $\mathsf{Ve}(\tilde{C}, \mathsf{en}) = 1$, and send key to V if so

   (e) Output the CDS value $z = \zeta \oplus \mathsf{RO}(\mathsf{key})$

4. V accepts if the correct key is received from P.

The proof of security for this protocol is essentially identical to that of Jawurek et al. [JKO13]. We give a sketch here for completeness.

Theorem 5.6.3. *Assuming $\mathcal{G}$ is a privacy-free garbling scheme and* CRHF *is a collision-resistant hash function, $\pi_{\mathsf{F}\cdot\mathbb{G}}$ UC-realizes $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ in the presence of an adversary statically corrupting up to one party, in the $\mathcal{F}^*_{\mathsf{COT}}$-hybrid local random oracle model.*

Proof. (Sketch) We describe how to simulate when the prover and verifier are corrupt as separate cases, and subsequently argue indistinguishability from the real execution.

Corrupt prover P^*. The prover P^* has little room to cheat. As the verifier has no private input, the simulator simply runs the verifier's code, with the only difference being that rather than accepting/rejecting based on whether P produces key alone, the simulator rejects any instance m, R^* where $R^* \neq \mathsf{F}_k(m) \cdot G$ (as per k received on behalf of $\mathcal{F}^*_{\mathsf{COT}}$). In order to

argue indistinguishability from the real protocol, it suffices to show that P^* is unable to derive key for any instance $R^* \neq \mathsf{F}_k(m) \cdot G$. Given an adversary P^* that claims at most N derived nonces, runs in time T, and succeeds in falsely proving at least one incorrect nonce with probability $\geq \epsilon$, we construct an efficient adversary for the authenticity property that succeeds with probability $\geq \epsilon/(NT)$. The reduction works as follows:

1. Receive k on behalf of $\mathcal{F}^*_{\mathsf{COT}}$, and sample $i \leftarrow [N]$
2. Run the code of honest V for every instance of nonce derivation except instance i
3. Compute the correct nonce for this i^{th} instance, $R_m = \mathsf{F}_k(m) \cdot G$
4. If the P^*'s claimed nonce for this instance $R^*_m = R_m$, then abort
5. If not, then send $\mathsf{F} \cdot \mathbb{G}, (k, m, R^*_m)$ to the authenticity challenger, and receive $\tilde{C}, \tilde{X}$ in response.
6. Sample $\zeta \leftarrow \{0,1\}^\kappa$, and send $\tilde{X}$ to P^* component wise on behalf of $\mathcal{F}^*_{\mathsf{COT}}$
7. Send $\tilde{C}, \zeta$ to P^* for the i^{th} instance on behalf of V
8. If P^* does not send key to V, upon P^* terminating (in which case $\hat{Y} = \mathsf{key}$), choose $\hat{Y}$ at random from the set of queries made by P^* to RO combined with the set of key values sent to the 'reveal' interface of $\mathcal{F}^*_{\mathsf{COT}}$
9. Send $\hat{Y}$ to the authenticity challenger and halt.

Note that the view of P^* in this reduction is the same as in the real protocol, up until the point that it queries key to RO or $\mathcal{F}^*_{\mathsf{COT}}$ for the i^{th} instance. Conditioned on the adversary having created a valid forgery (probability $\geq \epsilon$), the reduction finds the correct index for the forgery with probability $\geq 1/N$, and subsequently the correct query made to RO $/\mathcal{F}^*_{\mathsf{COT}}$ with probability $\geq 1/T$. Overall, the advantage of the reduction is therefore $\geq \epsilon/(NT)$. As $N, T \in \mathsf{poly}(\kappa)$ and $\mathcal{G}$ is authentic, ϵ must be negligible.

Corrupt verifier. The simulator for a corrupt verifier receives $(\mathsf{en}_{i,0}, \mathsf{en}_{i,1})_{i \in [\kappa]}$ on behalf of $\mathcal{F}^*_{\mathsf{COT}}$, extracts $\mathsf{de} = \mathsf{Ve}(\tilde{C}, \mathsf{en})$ (aborting if it fails). Finally it parses $(a, B) = \mathsf{de}$, computes $\mathsf{key} = a \cdot R + B$, and sends key to V iff it matches key^* received from V on behalf of $\mathcal{F}^*_{\mathsf{COT}}$ as the lock on the transmitted messages. Uniqueness of garbled outputs of the garbling scheme immediately gives us that the simulation is identical to the real protocol.

□

5.6.1 A Privacy Amplifying Optimization

While the ZKGC protocol makes only oracle use of $\mathcal{F}^*_{\mathsf{COT}}$, we can make an instantiation-specific optimization which will likely apply to any similarly structured instantiation, where the receiver only has statistical security inherited from statistical soundness of the decommitment/reveal phase. Currently, a naive instantiation would protect each choice bit of the receiver's (and hence private witness bit) with s bits of statistical security, i.e. there is at most a 2^{-s} probability of an adversary subverting the reveal phase by opening its message to a different one than committed earlier. As each instance of protocol π_{COT} makes use of independent randomness, the probability that a malicious sender is able to subvert the reveal phases of a *pair* of commitments is 2^{-2s}. Therefore if we are willing to tolerate one bit of leakage, we can in some sense consider the soundness of the reveal phase to be doubled.

Plugging the leak. The prover samples a random bit $r \leftarrow \{0,1\}$ during the one-time key setup phase. Now instead of directly using its witness bits $\mathbf{x}_i$ as the choice bit to the i^{th} instance of $\mathcal{F}^*_{\mathsf{COT}}$, the prover instead uses $\mathbf{x'}_i = \mathbf{x}_i \oplus r$ as the choice bit to the i^{th} instance. Finally the prover also inputs r as the choice bit to the $|\mathbf{x}| + 1^{\text{th}}$ instance of $\mathcal{F}^*_{\mathsf{COT}}$. When the time comes to evaluate a circuit $C(\mathbf{x})$, the prover and verifier instead use the circuit $C'(\mathbf{x'}, r) = C(\mathbf{x'}_1 \oplus r || \mathbf{x'}_1 \oplus r || \cdots || \mathbf{x'}_{|\mathbf{x}|} \oplus r)$ to cancel out the effect of r. Since XOR gates come for free in a garbled circuit [KS08], this adds essentially no overhead beyond the single extra instance of $\mathcal{F}^*_{\mathsf{COT}}$ for r. Now the input to C' can tolerate a single bit of leakage; any one bit leaked from $\mathbf{x'}||r$ is perfectly independent of $\mathbf{x}$.

Security. This clearly does not harm security against a corrupt prover, as the encoded input $\mathbf{x'}||r$ supplied to $\mathcal{F}^*_{\mathsf{COT}}$ unambiguously specify its candidate witness $\mathbf{x}$ just as earlier. As for when simulating for a corrupt verifier, in case one of the extractors for a π_{COT} instance i reports an extraction error for the key $k_{i,b}$ corresponding to bit b (this happens with probability 2^{-s}, but recall that the target security level is $2s$ bits) the simulator tosses a coin b'. If $b' = b$, then the simulator aborts the protocol (corresponding to a cheat being caught in the real protocol). Otherwise, the simulator simply runs the honest prover's protocol for the i^{th} bit going forward, effectively setting $\mathbf{x'}_i = \neg b$. The subtle point is that failing to extract $k_{i,b}$ does not hamper the simulator's ability to extract garbled circuits' embedded decoding information in the future: in case $i = |\mathbf{x}| + 1$, the value compromised is r, which does not influence any output wires anyway. In case $i \leq |\mathbf{x}|$, the simulator still obtains $k_{i,\neg b}$ by running the honest prover's code, and the availability of both keys on the r wire allow for the retrieval of both keys for $\mathbf{x}_i$ by evaluating the garbled circuit with $\mathbf{x'}_i \oplus 0$ and $\mathbf{x'}_i \oplus 1$ (i.e. substituting both values of r). Note that the evaluation will be 'correct' since the garbled

circuit is checked for correctness independently of $\mathcal{F}^*_{\mathsf{COT}}$.

Therefore in order to achieve $s' = 60$ bits of statistical security for ZKGC, one can parameterize the underlying OT protocol with $s = 30$ bits of soundness and remove the resulting leakage as described above.

5.6.2 Estimated Efficiency

We give estimates for an Ed25519 [BDL+12] style configuration. In particular, we assume a 256-bit curve, with SHA-512 as the PRF used to derive nonces just as in the EdDSA specification. SHA-512 has 58k AND gates [AMM+]. The nonce derivation key is 128 bits.

- **Garbled Circuit.** We can use a privacy-free garbled circuit in this context [FNO15], as the evaluator knows the entire input. In particular we can use the privacy-free variant of the Half Gates garbling scheme [ZRE15] which produces only one 128-bit ciphertext per AND gate. Each ciphertext is computed with two AES-128 invocations, and evaluated with one. The exponentiation gadget produces one 256-bit ciphertext for each output wire of the Boolean circuit. Consequently in the course of a single proof, V garbles the circuit (116k AES invocations), and P evaluates and verifies it (116k AES invocations). The bandwidth consumed is 928KB to transmit the garbled circuit $\tilde{C}$.

- $\mathcal{F}^*_{\mathsf{COT}}$**.** As discussed in Section 5.5.1, a single transfer and reveal instance costs 120 AES invocations and hashing a 1.92KB message via SHA-512 to compute, and 0.64KB in bandwidth. A single proof here requires 128 concurrent transfers and reveals via $\mathcal{F}^*_{\mathsf{COT}}$, bringing the computation cost to 16k AES invocations and hashing a 245KB message via SHA-512 for each P and V, and 81.92KB of bandwidth consumption.

Overall burden. In summary, P and V have roughly the same workload, dominated by 132k AES invocations and hashing a 245KB message. Each party additionally performs up to three curve multiplications, 256 additions in $\mathbb{Z}_q$, at little overhead. Bandwidth for $(|\tilde{C}| + |\mathcal{F}^*_{\mathsf{COT}}|)$ is 1.01MB.

The figures above are derived assuming SHA-512 is used for nonce derivation as is done by Ed25519, however it is likely that exploring standardized ciphers with smaller circuits such as AES will lead to substantial efficiency improvements. As an example, to derive a 256-bit nonce with bias $< 2^{-60}$ one could replace SHA-512 with three invocations of AES-128, which would incur a Boolean gate cost of $\approx 19k$ AND gates [AMM+]. This would bring the computation and bandwidth cost of $\tilde{C}$ down by a factor of 3, to $39k$ PRF calls and 307KB respectively. Note that in both scenarios (AES-128 and SHA-512), $\tilde{C}$ induces the dominant cost, as opposed to our $\mathcal{F}^*_{\mathsf{COT}}$ instantiation.

Given the cost breakup above, it is evident that the logistics for input encoding and exponentiation are no longer the bottleneck, and the cost of proving correctness of a derived nonce is now essentially dominated by the cost of garbling/evaluating and verifying the garbled circuit of the PRF (usually in the order of milliseconds [GLNP15, HK20]) used for nonce derivation.

5.7 Multiparty Dishonest Majority Threshold Signing

With the most complex component of stateless deterministic threshold signing - verifiable nonce derivation - instantiated, we are equipped to construct a clean multiparty signing protocol. The outline is as follows:

- **Setup:** All parties run canonical distributed key generation [Ped91] to obtain additive shares sk_i of a secret key sk (for public pk), and every pair of parties initializes an instance of $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ to commit to a nonce derivation key k_i. Note that we do not explicitly enforce any consistency across parties. Each party also samples a key k_* to derive randomness online.
- **Signing** m**:** Each party P_i derives its nonce $R_{m,i} = \mathsf{F}_{k_i}(m)$ and sends it to all other parties. Consistency is verified by standard echo-broadcast [GL05] in parallel with the next round. Every party derives its local random tape going forward by applying F_{k^*} on the digest of the view from the first round, i.e. $v = \mathsf{CRHF}(R_{m,0}||R_{m,1}||, \cdots ||R_{m,n})$. Each party P_i sets $(z_{i,j})_{j\in[n]\setminus i} = \mathsf{F}_{k^*}(j||v)$ and instructs $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ to deliver $z_{i,j}$ to P_j only if $R_{m,j}$ is the correct nonce. Finally each P_i sets the nonce to be $R_m = \sum_i R_{m,i}$ and computes its signature share

$$\sigma_i = (\mathsf{sk}_i \cdot H(\mathsf{pk}, R_m, m) + r_{m,i}) + \sum_{j\in[n]\setminus i} (z_{i,j} - z_{j,i})$$

  and sends it to all parties. The signature is then computed as $\sigma = \sum_i \sigma_i$.

Intuitively, P_i's share adds the mask $z_{i,j}$ to its contribution, and P_j's share removes this mask by subtracting $z_{i,j}$. Note that this is possible only if P_j obtained $z_{i,j}$ from $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ by having sent the correct $R_{m,j}$. Adding up all parties' σ_is cancels out the z values (if everyone is honest), and what remains is simply $\mathsf{sk} \cdot H(\mathsf{pk}, R_m, m) + r$ which is a valid signature on m.

We first give the exact functionality realized:

Functionality 5.7.1. $\mathcal{F}_{n,\mathsf{Sign}}$**.** n**-party Schnorr Signing**

This functionality is run among n parties $P_1, \cdots P_n$, and is parameterized by the group $(\mathbb{G}, G, q)$ and function $H : \{0,1\}^* \mapsto \mathbb{Z}_q$. All messages are adversarially delayed.

Key Generation: This phase is run exactly once, and must be run before any subse-

quent requests.

1. Wait to receive ($\texttt{init}$) from all parties.
2. Sample secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q$ and set the shared public key $\mathsf{pk} = \mathsf{sk} \cdot G$
3. Send ($\texttt{public-key}, \mathsf{pk}$) to all parties
4. Initialize function $\mathsf{F} : \{0,1\}^* \mapsto \mathbb{Z}_q \cup \{\perp\}$ to be $\perp$ everywhere unless otherwise specified.

Sign: Upon receiving ($\texttt{sign}, m$) from all parties,

1. If $\mathsf{F}(m) = \perp$, then sample $\mathsf{F}(m) \leftarrow \mathbb{Z}_q$.
2. Compute $r_m = \mathsf{F}(m)$ and $R_m = r_m \cdot G$, and send ($\texttt{nonce}, m, R_m$) to all parties.
3. Upon receiving ($\texttt{proceed}, m$) from all parties, compute

$$\sigma_m = \mathsf{sk} \cdot H(\mathsf{pk}, R_m, m) + r_m$$

and send ($\texttt{signature}, m, (\sigma_m, R_m)$) to all parties.

We give the n-party threshold signing protocol below.

Protocol 5.7.2. $\pi_{n,\mathsf{Sign}}$. **n-party threshold Schnorr**

This protocol is parameterized by the security parameter κ, elliptic curve group $(\mathbb{G}, G, q)$ ($|q| = \kappa$), PRF $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^\kappa \mapsto \mathbb{Z}_q$, and n parties $(P_i)_{i \in [n]}$. This protocol makes use of the ideal oracles $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ and $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$, and collision resistant hash function $\mathsf{CRHF} : \{0,1\}^* \mapsto \{0,1\}^\kappa$. The hash function used by Schnorr's signature scheme is $H : \{0,1\}^* \mapsto \mathbb{Z}_q$.

Distributed Key Generation: Each P_i does the following:

1. Sample nonce derivation key $k_i \leftarrow \{0,1\}^\kappa$
2. Sample secret sharing randomness key $k_* \leftarrow \{0,1\}^\kappa$
3. For each $j \in [n] \setminus i$, $sid_{i,j}$ will be used for an instance of $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ where P_i is the prover, and P_j is the verifier. Send ($sid_{i,j}, \texttt{input-key}, k_i$) to $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ and wait for ($sid_{j,i}, \texttt{initialized}$) in response.
4. Sample secret key share $\mathsf{sk}_i \leftarrow \mathbb{Z}_q$ and set public key share $\mathsf{pk}_i = \mathsf{sk}_i \cdot G$

5. Send $(\texttt{commit}, i, \mathsf{pk}_i, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{Com-ZK}}^{\mathsf{R}_{DL}}$
6. Upon receiving $(\texttt{committed}, j)$ from $\mathcal{F}_{\mathsf{Com-ZK}}^{\mathsf{R}_{DL}}$ for each $j \in [n] \setminus i$, send $(\texttt{reveal})$ to $\mathcal{F}_{\mathsf{Com-ZK}}^{\mathsf{R}_{DL}}$
7. Wait to receive $(\texttt{revealed}, \mathsf{pk}_j)$ from $\mathcal{F}_{\mathsf{Com-ZK}}^{\mathsf{R}_{DL}}$ for each $j \in [n] \setminus i$
8. Assemble the shared public key $\mathsf{pk} = \sum_{i \in [n]} \mathsf{pk}_i$

Sign: With common input $msg \in \{0,1\}^*$, each P_i does the following:

- **Round 1**:
  1. Set $m = \mathsf{CRHF}(msg)$
  2. Derive nonce share $r_{m,i} = \mathsf{F}_{k_i}(m)$ and set $R_{m,i} = r_{m,i} \cdot G$
  3. Send $R_{m,i}$ to each P_j for $j \in [n] \setminus i$
  4. Wait to receive $R_{m,j}$ from each P_j for $j \in [n] \setminus i$
- **Round 2**:
  5. Compute digest of previous round, $\mathsf{v}_i = \mathsf{CRHF}(R_{m,1}, R_{m,2}, \cdots, R_{m,n})$
  6. For each $j \in [n]$, generate CDS value $z_{i,j} = \mathsf{F}_{k^*}(j || \mathsf{v}_i)$ and conditionally disclose it to P_j by sending $(sid_{j,i}, \texttt{verify-nonce}, m, R^*_{m,j}, z_{i,j})$ to $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$. Prove own nonce by sending $(sid_{i,j}, \texttt{verify-nonce}, m, R^*_{m,i})$ to $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$.
  7. Send v_i to all parties
  8. For each $j \in [n]$, wait to receive v_j from P_j and $(sid_{j,i}, \texttt{secret}, m, z_{j,i})$ from $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$
- **Round 3**:
  9. Abort if a single $\mathsf{v}_j \neq \mathsf{v}_i$
  10. Assemble shared nonce $R_m = \sum_{i \in [n]} R_{m,i}$
  11. Compute signature share such that it masks/adds every CDS value generated locally, and unmasks/removes every CDS value received from other parties:

$$\sigma_i = (\mathsf{sk}_i \cdot H(\mathsf{pk}, R_m, m) + r_{m,i}) + \sum_{j \in [n] \setminus i} (z_{i,j} - z_{j,i})$$

  12. Send σ_i to all parties

13. For each $j \in [n] \setminus i$, wait to receive σ_j from P_j and $(sid_{j,i}, \texttt{verified}, m, R^*_{m,j})$ from $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$
14. Assemble the signature $\sigma = \sum_{j\in[n]} \sigma_j$
15. If (R_m, σ) is not a valid signature on msg, or $(sid_{j,i}, \texttt{fail}, m, R^*_{m,j})$ is received from $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ for any $j \in [n]$ then abort.
16. Otherwise, output (R_m, σ)

Theorem 5.7.3. *Assuming* F *is a pseudorandom function and* CRHF *is a collision-resistant hash function,* $\pi_{n,\mathsf{Sign}}$ *UC-realizes* $\mathcal{F}_{n,\mathsf{Sign}}$ *in the presence of an adversary statically corrupting up to* $n-1$ *parties, in the* $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$,$\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$*-hybrid model.*

Proof. (Sketch) Simulating this protocol for an adversary corrupting $n-1$ parties is done as follows: let the honest party by indexed by $j \in [n]$. The simulator receives k_i on behalf of $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ and sk_i on behalf of $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$ from each corrupt P_i, and upon receiving $\mathsf{pk}$from $\mathcal{F}_{n,\mathsf{Sign}}$, reveals $\mathsf{pk}_j = \mathsf{pk} - \sum_{i\in[n]\setminus j} \mathsf{pk}_i$ to corrupt parties on behalf of $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$. When signing a message msg, with $m = \mathsf{CRHF}(msg)$ the simulator first receives nonce R_m from $\mathcal{F}_{n,\mathsf{Sign}}$, and sends $R_{m,j} = R_m - (\sum_{i\in[n]\setminus j} \mathsf{F}_{k_i}(m)) \cdot G$ to all parties on behalf of P_j. On receiving $R_{m,i}$ from each P_i, if this exact set of $R_{m,i}$ values has not previously been seen, the simulator samples a fresh $(z_{j,i})_{i\in[n]\setminus j} \leftarrow \mathbb{Z}_q^{n-1}$ (otherwise reuses these values from the last time). For each $i \in [n] \setminus j$, if $R_{m,i} = \mathsf{F}_{k_i}(m) \cdot G$ the simulator sends $z_{j,i}$ to P_i on behalf of $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$. If even one of the $R_{m,i}$ values is incorrect, the simulator samples a uniform $\sigma_j \leftarrow \mathbb{Z}_q$ and sends it to each P_i and aborts. Otherwise, the simulator asks $\mathcal{F}_{n,\mathsf{Sign}}$ for a signature, receiving σ in response, and computes σ_j as follows:

$$\sigma_j = \sigma - \sum_{i\in[n]\setminus j} (\mathsf{sk}_i \cdot H(\mathsf{pk}, R_m, m) + \mathsf{F}_{k_i}(m)) + \sum_{i\in[n]\setminus j} (z_{j,i} - z_{i,j})$$

where $z_{i,j}$ is received from P_i on behalf of $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$. The simulator sends σ_j to all parties on behalf of P_j.

Indistinguishability of the simulation is argued as follows: first, by collision resistance of CRHF, no two messages or sets $\{R_{m,i}\}$ induce the same random tape for P_j. Second, as F is a PRF, the pseudorandom $R_{m,j}$ values in the real protocol are computationally indistinguishable from their uniformly random counterparts in the simulation. The same holds for $z_{j,i}$ values. Finally the only non-syntactic difference between the simulation and the real protocol is that when $\exists i \in [n]$ such that $R_{m,i} \neq \mathsf{F}_{k_i}(m) \cdot G$, the simulator produces a uniformly random σ_j instead of computing it as a function of σ received from $\mathcal{F}_{n,\mathsf{Sign}}$. This induces no change in the view of the adversary, as in the real protocol $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ withholds $z_{j,i}$

from P_i in this event (so the adversary has no information about this value in its view), and so $z_{j,i}$ acts as a one-time pad within σ_j in the real protocol.

Additionally in the event that there is more than one honest party, collision-resistance of CRHF immediately guarantees that no two parties will have an inconsistent view of $\{R_{m,i}\}$ - any inconsistency will induce honest parties to abort before they reveal any information about σ in the final round. □

5.7.1 Efficiency

The protocol is essentially a thin wrapper on top of $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$, and consequently the cost is dominated by running $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ between every pair of parties. Every pair of parties P_i, P_j shares two instantiations of $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$, one in which P_i plays the prover and $P_j$the verifier, and another with the roles reversed. However by the structure of our protocol $\pi_{\mathsf{F}\cdot\mathbb{G}}$, instantiating $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ in both directions induces little computational overhead on top of a single instantiation: while the verifier garbles the circuit the prover sits idle, and while the prover evaluates the garbled circuit the verifier has nothing to do. This means that when P_i is working as the verifier in one instance of $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ with P_j, it will be idling in its role as the prover in the other instance of $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ with P_j, and vice versa.

For this reason, we expect a two-party instantiation of $\pi_{n,\mathsf{Sign}}$ to incur the same bandwidth and computation cost per party as calculated in Section 5.6.2. This cost is multiplied by n for an n party instantiation.

5.8 Open Problems

The goal of our work has been to a construct computationally light mechanism to realize $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$. We explored the ZKGC paradigm in particular for this task, and developed new techniques to suppress the 'logistical' costs associated with the proof, so that the garbled circuit itself is the heaviest component. There is certainly scope to improve upon our techniques even within the ZKGC paradigm—in particular, the committed OT realization likely has room to improve efficiency.

Beyond the ZKGC paradigm, it might be interesting to explore the tradeoffs induced by different zero-knowledge proof techniques to instantiate $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$. For instance, the MPC-in-the-head paradigm [IKOS07, BHH+19] could also be conducive to a computationally light approach. In settings where bandwidth proves to be more of a constraint than computational resources, one could explore the many succinct proof systems available [BCR+19, BBB+18, Gro16] that offer various tradeoffs in bandwidth and computation cost.

Chapter 6

Proactive Threshold Wallets with Offline Devices

In this chapter, we begin by carefully formulating a meaningful notion of offline refresh for a (t, n) cryptosystem, with discussions about tradeoffs, impossibilities, and instantiations, and then incrementally build our techniques so that we may finally construct a $(2, n)$ threshold ECDSA construction that achieves offline refresh. We give empirical justification of its practicality. Finally, we prove that this notion of offline refresh is too strong to satisfy with a dishonest majority when $t > 2$.

6.1 Defining Offline Refresh

A notion of offline refresh that is not a priori too restrictive or offers too weak a security guarantee is tricky to define. Existing definitions (eg. [ADN06]) require that the refresh procedure always terminate successfully when honest parties receive the instruction. This can be viewed as the proactive analog of the well-studied MPC notion of Guaranteed Output Delivery (GOD). It is immediate from foundational results on dishonest majority coin tossing [Cle86] that if there is no honest majority involved in the refresh procedure that achieves GOD, then the resulting randomness for proactivization is succeptible to unacceptable bias.

One may consider instead a proactive analog of the MPC notion of security with abort. This notion allows the adversary to abort the computation if it so desires, possibly receiving output while depriving honest parties of it. Efficient dishonest majority MPC protocols that achieve security with abort are known in the literature [DPSZ12, KOS16] indicating that this notion may be the correct one.

However one must be careful when defining exactly what power to allow the adversary in

aborting the refresh procedure. Security with abort in the standard MPC setting comes with a fine-grained separation between *selective* and *unanimous* abort [FGH+02], the difference being that in the former some honest parties may get output while others not, while in the latter all honest parties agree on whether or not to abort. In standard MPC protocol design the choice between these two security notions offers a meaningful tradeoff: selective abort while offering strictly weaker security is sufficient for many applications, and is much more efficient in round complexity and/or use of broadcast [GL05]. When translated to the setting of proactive security however we argue that this distinction is much more drastic, to the point of making selective abort patently undesirable.

Refresh with selective abort is insufficient Consider the following adaptation of security with selective abort: at the end of the refresh protocol, the adversary has the power to choose exactly which (honest) parties successfully advance to the next epoch. This gives the adversary the power to execute attacks on the honest parties' private state that were not feasible without the proactivization protocol. In particular an adversary could for instance convince one half of the honest parties to advance to the next epoch while the remaining honest parties do not. As the parties that advance erase their state from the previous epoch, their secrets will no longer be correlated with the parties that do not advance. This means that even if the system has an honest majority of parties (which in the static setting means the shared secret can always be reconstructed/used if desired), the refresh procedure gives the adversary a window to throw the parties out of sync and 'erase' the common secret from the system's distributed state.

Concretely this could translate to attacks where a single malformed message or network issue causes a threshold wallet to permanently erase the common secret key, which in many cases could mean an irreversible loss of funds.

Refresh with unanimous erasure We settle on 'unanimous erasure' as the correct definition for proactive security, as the analog of security with unanimous abort. Informally, this means that the adversary has the power to decide whether or not to move to the next epoch, but crucially *all honest parties agree on the epoch* with the caveat that they may not be activated synchronously. Offline refresh is captured by allowing the adversary to advance the epoch arbitrarily many times (and even change corruptions) without activating *all* honest parties, however any honest party if activated must 'catch up' non-interactively to the current epoch.

Corruption Caveats Defining a meaningful model that allows different parties to stay "offline" (and therefore effectively exist in different epochs at the same time) while simultaneously honouring the assumption that only a threshold number of parties are corrupt at any given epoch requires particular care. We handle this issue by requiring that the adversary allows a party to "update" before corrupting it. While this appears to weaken the model, a definition without this restriction would be inherently unachievable, as an adversary would be able to effectively "travel in time". For instance, if some party P is offline from epoch i onward, an adversary who corrupts it after the the system has progressed to epoch $i+1$ will obtain this party's state at epoch i even after that epoch has passed. This would be problematic if the adversary had already corrupted (and subsequently uncorrupted) $t-1$ different parties at epoch i, as gaining P's state for epoch i will completely reveal the system's secrets, all without violating the assumption that only $t-1$ parties may be corrupt at any given point in time. See the paragraph on *Corruptions* in the formal definition that follows for further discussion.

Parameters The system consists of n parties, of which t are necessary to `operate` by accessing the secret. The adversary may corrupt at most $t-1$ parties. The `refresh` procedure is run by activating t_ρ parties.

With these security notions in mind, we formalize the definition of proactive security with unanimous erasure and offline refresh in the UC model below.

We build on the definition of Almansa et al. [ADN06] to a notion of mobile adversaries that accommodates 'offline' parties. We do this by having each party maintain a counter epoch written on a special tape, and define the state of the system relative to these epoch values. While in our definition the adversary $\mathcal{Z}$ may choose to activate parties in sequences that leave them in different epochs, the definition of Almansa et al. does not permit this. In particular their definition requires all honest parties to first agree that they have all successfully reached the latest epoch before the adversary is permitted to change corruptions.

Epochs Each party has a special "epoch tape" on which it writes an integer epoch. At the start of the protocol, this tape contains the value 0 for all honest parties. We use the term "system epoch" to refer to the largest epoch value written on any honest party's tape.

Operations There are two kinds of commands that the environment $\mathcal{Z}$ can send to a party: `operate`, `refresh`, and `update`. Intuitively `operate` corresponds to use of the system's service, `refresh` the candidate proactivization generation, and `update` the application of this proactivization to rerandomize parties' private state. The `operate` command will be issued

to t parties simultaneously (in any realization this will require them to interact), `refresh` to t_ρ parties (also requiring interaction), and `operate` will be individual and non-interactive in its realization.

Non-degeneracy Upon being given the `refresh` command, an honest party must write the current system epoch on its epoch tape. In order to rule out degenerate realizations, we also require that if any t honest parties are given the `operate` command, the next `refresh` command sent to an honest party P will result in the system epoch being incremented.

Corruptions At any given time, there can be at most $t-1$ parties controlled by $\mathcal{Z}$. Mobility of corruptions must adhere to the following rule: $\mathcal{Z}$ may decide to "uncorrupt" a party P at any time, however before corrupting a new party $P' \in \mathbf{P}$ it must first "leave" P, then send `refresh` to any t_ρ parties without aborting (i.e. increments the epoch counter), and finally `update` to P' before being given its internal state (and full control over subsequent actions). Note that omitting this final `update` message (i.e. allowing $\mathcal{Z}$ to corrupt P' before it has refreshed) will give $\mathcal{Z}$ the views of both P and P' from the same system epoch, in which case the system will be fully compromised. This is implied by any standard definition of proactive security. In fact, our revised definition grants $\mathcal{Z}$ more power than that of Almansa et al. [ADN06], as here not every party need refresh before $\mathcal{Z}$ changes corruptions.

Crucially we allow the system epoch to be pushed forward by *any* t_ρ *parties*, i.e. consecutive epoch increments may be enabled by completely non-overlapping sets of parties. This captures our notion of "offline refresh" where not all parties in the system need be online to move the system forward; any t_ρ parties can keep the epoch counter progressing while the others catch up at their own speed.

Offline-refresh must be non-interactive A direct implication of our definition is that one can not wait for offline parties to respond before incrementing the epoch counter. This inherently rules out standard interactive verifiable secret sharing (VSS) approaches where parties 'complain' if an adversary tries to cheat them. Previous proactive secret sharing protocols can be viewed as implementing such a VSS between epochs (either explicitly by complaints against misbehaviour, or implicitly by voting for 'good' sharings), and so a fundamentally different approach is required for the offline-refresh setting.

6.2 Instantiating Offline Refresh

With the model and definitions in place, we now incrementally work towards our protocol via a sequence of stepping stones to introduce which tools we use and why.

6.2.1 Simple Honest Majority Instantiation

We begin by sketching a 'baby protocol' for proactive secret sharing with $t_\rho = 2t - 1$ and $n = t_\rho + 1$, i.e. where the refresh protocol is run by an honest majority of online parties and one party (labelled P_{off}) stays offline.

Network It is immediate that a necessary underlying assumption is a forward secure channel that supports delivery to offline parties. Formally, this is captured by having offline parties accumulate messages in a buffer that they read when they become online. In practice an offline party may not literally be disconnected from the network and need a buffer, just that the refresh protocol does not require its participation. Alternatively message delivery may be aided by a server as in the Signal protocol [MP, ACD19]. We assume that the t_ρ online parties share a broadcast channel (which is not necessarily visible to P_{off}).

Cryptographic tools As a parameter of the protocol, parties agree on an elliptic curve $\mathbb{G}$ generated by G and of order q, where the Discrete Logarithm problem is assumed to be hard. We assume two protocol primitives:

- $\pi^{\mathsf{DKG}}_{\mathsf{Setup}}$ is a protocol where at the end each party P_j holds $\mathsf{sk}_j = f(j) \in \mathbb{Z}_q$ where f is a degree $t-1$ polynomial with the common secret defined as $\mathsf{sk} = f(0)$. Additionally every party knows $\mathsf{pk}_j = F(j) = f(j) \cdot G$ for each $j \in [n]$. This is a common tool [Fel87b, Ped92] and we recall a canonical instantiation in Appendix C.1.
- $\mathsf{Reshare}(i)$ is a protocol run by t_ρ parties each of whom have local secret shares $f(j)$ and public shares $F(j)$ as created by $\pi^{\mathsf{DKG}}_{\mathsf{Setup}}$ above, in order to create a fresh and independent sharing of the same format where the secret is $f(i)$. In particular, at the end of $\mathsf{Reshare}(i)$, each party P_j holds $f'(j) \in \mathbb{Z}_q$ where f' is a degree $t-1$ polynomial with $f'(0) = f(i)$. This is a common tool as well, and so we refer the reader to Gennaro et al. [GRR98] for further details.

As before, let $\mathsf{off} = t_\rho + 1$ index the offline party. The refresh protocol is run among t_ρ online parties as follows:

1. Parties $P_1, \cdots P_{t_\rho}$ run $\mathsf{Reshare}(0)$ in order to obtain fresh shares the secret key, i.e. they agree on a public degree $t-1$ polynomial F over $\mathbb{G}$ and each P_j obtains $f(j)$ such that $f(j) \cdot G = F(j)$. It holds that $F(0) = \mathsf{pk}$. They overwrite $\mathsf{sk}_j = f(j)$ and $\mathsf{pk}_j = F(j)$ for each $j \in [n]$.

2. They then run $\mathsf{Reshare}(\mathsf{off})$ to jointly sample a fresh degree $t-1$ polynomial f' such that $f'(0) = \mathsf{sk}_{\mathsf{off}}$ and each P_j knows $f'(j)$ and every public $F'(j) = f'(j) \cdot G$.

3. Each P_j for $j \in [t_\rho]$ sends $\mathsf{pk} = (\mathsf{pk}_j)_{j \in [t_\rho]}, f'(j), F'$ privately to P_{off}.

It holds that since there are t honest parties who execute the final step, upon waking up P_{off} will find at least t messages that agree on pk, F' accompanied by as many correct evaluations $f'(j)$ which can be verified by checking $f'(j) \cdot G \stackrel{?}{=} F'(j)$. Note that since there are at most $t-1$ malicious parties, they can't collude to create a sufficiently large set to fool P_{off}. It is immediate that P_{off} can therefore interpolate the correct $\mathsf{sk}_{\mathsf{off}}$ and 'catch up' on all the refreshes that it missed. This protocol can easily be extended for an arbitrary number of offline parties by generating a new reshared polynomial for each of them.

Hence we have shown that offline refresh is easy to satisfy in the presence of an online honest majority.

6.2.2 Dishonest Majority with Offline Broadcast

Folklore techniques such as Cleve [Cle86] give strong evidence that unanimous erasure in a $(2, 3)$ system is impossible to achieve over private channels alone. We give a rough sketch here as to why this is the case.

Consider a system comprising $P_0, P_1, P_{\mathsf{off}}$ in which P_{off} is offline, one of P_0 or P_1 may be corrupt, and the honest party and P_{off} must either agree on a random bit (successful termination) or agree to abort. The non-degeneracy requirement is that an honest execution does not induce an abort. Additionally the parties have access to arbitrary correlated randomness generated in some offline phase, which rules out direct application of the $t < n/3$ consensus lower bound [PSL80]. This system and its constraints captures a simplified notion of unanimous erasure.

We will argue that if P_0 is corrupt, then P_1 and P_{off} can not meet the constraints of the system. Observe that in the event of successful termination the private communication from P_0 to P_{off} is by itself sufficient to 'convince' P_{off} not to abort; if this were not true then a corrupt P_1 could simply erase its entire private channel, which forces P_{off} to abort while honest P_0 who is unaware of this terminates successfully. We call a transcript from either one of P_0 or P_1 to P_{off} as 'convincing' if it induces P_{off} to terminate successfully with an

output bit instead of aborting. Without loss of generality there must be some round in the protocol where P_0 gains the ability to produce a convincing transcript, but P_1 has not yet acquired this ability (either party having this ability from round 0 would clearly admit trivial attacks). Therefore if P_0 simply halts the protocol with P_1 at this point, P_1 will have no way of knowing whether P_0 will choose to convince P_{off} to abort or to terminate successfully.

Offline Broadcast In order to overcome this challenge we introduce a powerful notion of an 'offline broadcast channel', which is a broadcast channel shared by $P_0, P_1, P_{\mathsf{off}}$ but crucially is invisible to the adversary if none of the parties are corrupt. Our final protocol will not use so strong a tool, but it provides an instructive stepping stone.

Leaking the Difference Polynomial We observe that *any* proactivization protocol where an adversary corrupts t parties has the following property: define $f_\delta(i) = f'(i) - f(i)$, i.e. the polynomial that encodes the difference between old and new shares. Given $f(i), f'(i)$ for any $t-1$ values of i (which the adversary has by virtue of corrupting $t-1$ parties) one can compute $f_\delta(x)$ for any x. This is because $f_\delta(0) = 0$ (as $f(0) = f'(0)$) and f_δ is a degree $t-1$ polynomial of which one now has t points.

Given an offline broadcast channel, designing a refresh protocol for $P_0, P_1, P_{\mathsf{off}}$ using the above observation is as simple as sampling the difference polynomial on the broadcast channel. In particular the refresh protocol proceeds as follows:

1. P_0 samples a uniform $f_{\delta,0}$ and offline-broadcasts a commitment to $f_{\delta,0}$.
2. P_1 samples a uniform $f_{\delta,1}$ and offline-broadcasts it.
3. P_0 decommits f_δ on the offline-broadcast channel.
4. Each party (either immediately, or upon waking up) defines $f_\delta = f_{\delta,0} + f_{\delta,1}$ and updates its local share as $f'(i) = f(i) + f_\delta(i)$

It is clear that the above offline refresh protocol tolerates a mobile malicious adversary that corrupts at most one party at any given time (which is optimal in a $t = 2$ system). In particular the offline broadcast channel allows for the following properties:

- The online parties and P_{off} use the same criteria to compute f_δ and so are always in agreement.
- Since the offline broadcast channel is invisible to the adversary when switching corruptions, the uniform choice of f_δ ensures that the resulting refreshed polynomial is distributed independently of any parties' view from earlier.

Unfortunately this offline broadcast primitive is an unreasonably strong assumption to make in practice. Broadcast is either implemented via interactive protocols, or inherently public when using a ledger/blockchain. We therefore carefully design a protocol that somewhat achieves the effect of this offline broadcast channel; we will use *private channels to communicate candidate f_δ values along with a public ledger to reach agreement on whether or not to use them*, and rely on the intrinsic entropy of certain common threshold signatures to bind the public and private components.

A Note on Parameters As our subsequent constructions are explicitly for $t = t_\rho = 2$, we drop the t, t_ρ notation until we revisit the general multiparty setting in Section 6.8.

6.3 Threshold Signature Abstraction

A threshold signature scheme [Des88] allows the power of producing a digital signature to be delegated to multiple parties, so that a threshold number of them must work together in order to produce a signature. Specifically a (t, n) signature scheme is a system in which n parties hold shares of the signing key, of which any t must collaborate to sign a message. In this section we focus on $(2, n)$ threshold versions of the ECDSA [Kra93] and Schnorr [Sch91] Signature schemes. As our techniques are general and not specific to any one threshold signature scheme, we use an abstraction of such protocols for ease of exposition.

6.3.1 Abstraction

We assume that a $(2, n)$ threshold signature over group $(\mathbb{G}, G, q)$ can be decomposed in a triple of algorithms $(\pi^{\mathsf{DKG}}_{\mathsf{Setup}}, \pi^{\mathsf{R}}_{\mathsf{Sign}}, \pi^{\sigma}_{\mathsf{Sign}})$ of the following formats:

- $(\mathsf{sk}_i \in \mathbb{Z}_q,\ \mathsf{pk} \in \mathbb{G}) \leftarrow \pi^{\mathsf{DKG}}_{\mathsf{Setup}}(\kappa)$
  This protocol is run with n parties and has each honest party P_i obtain public output pk and private output sk_i. In addition to this, there must exist a degree-1 polynomial f over $\mathbb{Z}_q$ such that $\forall i \in [n]$, $\mathsf{sk}_i = f(i)$.

- $(R \in \mathbb{G},\ \mathsf{state}_b \in \{0,1\}^*) \leftarrow \pi^{\mathsf{R}}_{\mathsf{Sign}}(\mathsf{pk}, \mathsf{sk}_b, 1-b, m)$
  Run by party P_b with P_{1-b} as counterparty, to sign message m. Both parties output the same R when honest, with private state state_b.

- $(\sigma \in \mathbb{Z}_q) \leftarrow \pi^{\sigma}_{\mathsf{Sign}}(\mathsf{state}_b)$
  Completes the signature started by $\pi^{\mathsf{R}}_{\mathsf{Sign}}$ when both parties are honest, i.e. σ verifies as a signature on message m with R as the public nonce and pk as the public key.

Note that $\pi^{\mathsf{DKG}}_{\mathsf{Setup}}$ captures a specific kind of secret sharing, i.e. the kind where the signing key is Shamir-shared among the parties. Multiplicative shares for instance are not captured by this abstraction. The (2,2) threshold ECDSA protocols of Lindell [Lin17] and Castagnos et al. [CCL+19] are not captured by our abstraction for this reason. Additionally signature schemes that do not have randomized signing algorithms such as BLS [BLS01] can not be decomposed as per this abstraction.

Finally these protocols must realize the relevant threshold signature functionality. In particular let $\mathsf{Sign} \in \{\mathsf{Sign}^H_{\text{ECDSA}}, \mathsf{Sign}^H_{\text{Schnorr}}\}$ where

$$\mathsf{Sign}^H_{\text{ECDSA}}(\mathsf{sk}, k, m) = \frac{H(m) + \mathsf{sk} \cdot r_x}{k}$$
$$\mathsf{Sign}^H_{\text{Schnorr}}(\mathsf{sk}, k, m) = H(R||m) \cdot \mathsf{sk} + k$$

where r_x is the x-coordinate of $k \cdot G$ in the ECDSA signing equation. We therefore define functionality $\mathcal{F}^{n,2}_{\mathsf{Sign}}$ to work as follows:

Functionality 6.3.1.

Discrete Log Based Threshold Signature $\left(\mathcal{F}^{n,2}_{\mathsf{Sign}}\right)$ This functionality is parameterized by the party count n, the elliptic curve $(\mathbb{G}, G, q)$, a hash function H, and a signing algorithm Sign. The setup phase runs once with n parties, and the signing phase may be run many times between (varying) subgroups of parties indexed by $i, j \in [n]$.

Setup On receiving (`init`) from all parties,

1. Sample and store the joint secret key,

$$\mathsf{sk} \leftarrow \mathbb{Z}_q$$

2. Compute and store the joint public key,

$$\mathsf{pk} := \mathsf{sk} \cdot G$$

3. Send (`public-key`, pk) to all parties.

4. Store (`ready`) in memory.

Signing On receiving $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}}, (i,j), m)$ from both parties indexed by $i, j \in [n]$ $(i \neq j)$, if $(\texttt{ready})$ exists in memory but $(\texttt{complete}, \mathsf{id}^{\mathsf{sig}})$ does not exist in memory, then

1. Sample $k \leftarrow \mathbb{Z}_q$ and store it as the instance key.
2. Wait for $(\texttt{get-instance-key}, \mathsf{id}^{\mathsf{sig}})$ from both parties P_i, P_j.
3. Compute
$$R := k \cdot G$$
and send $(\texttt{instance-key}, \mathsf{id}^{\mathsf{sig}}, R)$ to parties P_i, P_j. Let $(r_x, r_y) = R$.
4. Wait for $(\texttt{proceed}, \mathsf{id}^{\mathsf{sig}})$ from both parties P_i, P_j.
5. Compute
$$\sigma := \mathsf{Sign}^H(\mathsf{sk}, k, m)$$
6. Send $(\texttt{signature}, \mathsf{id}^{\mathsf{sig}}, \sigma)$ to both parties P_i, P_j as adversarially-delayed private output.
7. Store $(\texttt{complete}, \mathsf{id}^{\mathsf{sig}})$ in memory.

To make concrete the role of each protocol $(\pi^{\mathsf{DKG}}_{\mathsf{Setup}}, \pi^{\mathsf{R}}_{\mathsf{Sign}}, \pi^{\sigma}_{\mathsf{Sign}})$, we restrict access of their corresponding simulators $(\mathcal{S}^{\mathsf{DKG}}_{\mathsf{Setup}}, \mathcal{S}^{\mathsf{R}}_{\mathsf{Sign}}, \mathcal{S}^{\sigma}_{\mathsf{Sign}})$ to $\mathcal{F}^{n,2}_{\mathsf{Sign}}$. Specifically $\mathcal{S}^{\mathsf{DKG}}_{\mathsf{Setup}}$ can only send $(\texttt{init})$ on behalf of a corrupt party and receive $(\texttt{public-key}, \mathsf{pk})$ in response. The messages $(\texttt{sign}, \mathsf{id}^{\mathsf{sig}}, (i,j), m)$ and $(\texttt{get-instance-key}, \mathsf{id}^{\mathsf{sig}})$ can be sent and $(\texttt{instance-key}, \mathsf{id}^{\mathsf{sig}}, R)$ received only by $\mathcal{S}^{\mathsf{R}}_{\mathsf{Sign}}$. Finally $(\texttt{proceed}, \mathsf{id}^{\mathsf{sig}})$ can be sent and $(\texttt{signature}, \mathsf{id}^{\mathsf{sig}}, \sigma)$ received only by $\mathcal{S}^{\sigma}_{\mathsf{Sign}}$.

An implication of this restriction is that $\pi^{\mathsf{R}}_{\mathsf{Sign}}$ has to be simulatable without the signature σ, therefore it cannot leak any information about this value. (The approach of splitting the simulator into several simulators to limit what kind of information can be leaked in different stages of the protocol has been used before e.g., in secret-sharing based MPC protocols to claim that the protocol does not leak any information about the output until the reconstruction phase performed in the last round of the protocol). This abstraction was chosen deliberately to enforce this property; one of our key techniques in this work (Section 6.5) relies on $\pi^{\mathsf{R}}_{\mathsf{Sign}}$ keeping σ hidden.

Threshold Schnorr We recall a folklore instantiation of $\mathcal{F}^{n,2}_{\mathsf{Sign}}$ for $\mathsf{Sign}_{\mathrm{Schnorr}}$ in Appendix C.1 (note that this also works for EdDSA).

Threshold ECDSA We note that the recent protocols of Gennaro and Goldfeder [GG18], Lindell et al. [LNR18], and Doerner et al. [DKLs19] for $\mathsf{Sign}_{\mathrm{ECDSA}}$ can also be cast in the above framework if required. However due to the non-linearity of $\mathsf{Sign}_{\mathrm{ECDSA}}$ the corresponding realization of $\mathcal{F}^{n,2}_{\mathsf{ECDSA}}$ requires use of a multiplication functionality $\mathcal{F}_{\mathsf{MUL}}$ (or equivalent protocol). Since $\mathcal{F}_{\mathsf{MUL}}$ is expensive to instantiate for one-time use, these threshold ECDSA protocols run some preprocessing for $\mathcal{F}_{\mathsf{MUL}}$ in parallel with $\pi^{\mathsf{DKG}}_{\mathsf{Setup}}$ and make use of this preprocessed state for more efficient online computation. As this adds additional persistent state to be protected against a mobile adversary, we need to deal with it carefully. We discuss this in further detail and give an efficient solution to this problem in Section 6.6.

6.4 Coordinating Two Party Refresh

As the final protocol combines two independent concepts: using the blockchain for synchronization, and authenticating communication to offline parties, we first present a base protocol for the former for a $(2, 2)$ access structure and augment it with the latter to obtain a $(2, n)$ protocol. In this section, we describe the malicious secure protocol for two parties to coordinate an authenticated refresh of the secret key shares. The $(2, 2)$ protocol is described with Shamir secret shares (points on a polynomial) rather than just additive shares so as to allow for a smoother transition to the $(2, n)$ setting.

Intuition The two parties begin by running the first half of the threshold signing protocol $\pi^{\mathsf{R}}_{\mathsf{Sign}}$ to obtain the signing nonce R that will be used for the subsequent threshold signature itself. They then sample a new candidate (shared) polynomial f' by publicly sampling the difference polynomial f_δ and store their local share $\mathsf{sk}'_b = f'(b)$ tagged with R and the epoch number epoch in a list rpool. Specifically rpool is a list of $(R, \mathsf{sk}'_b, \mathsf{epoch})$ values that are indexed by R as the unique identifying element. Following this, they complete the threshold signing by running $\pi^{\sigma}_{\mathsf{Sign}}$ and a designated party sends the resulting signature (and message) to $\mathcal{G}_{\mathsf{Ledger}}$, i.e. posts them to the public ledger.

Protocol 6.4.1. $\pi^{(2,2)}_{\rho\text{-}\mathsf{sign}}$**.** $(2, 2)$ **Schnorr Signing With Refreshment**

Parameters: Elliptic Curve Group $(\mathbb{G}, G, q)$

Parties: P_b, P_{1-b} (recall $b \in \{1, 2\}$ is the index of the current party and $1 - b$ is a shorthand for the index of the counterparty)

Ideal Oracles: $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$, $\mathcal{G}_{\mathsf{Ledger}}$

Inputs:

- **Common**: Message to be signed $m \in \{0,1\}^*$, public key $\mathsf{pk} \in \mathbb{G}$, each party's share in the exponent $\mathsf{pk}_b = \lambda_b^{1-b}(0) \cdot F(b)$ where F is the polynomial over $\mathbb{G}$ passing through $(0, \mathsf{pk})$ and $(b, f(b) \cdot G)$, epoch index $\mathsf{epoch} \in \mathbb{Z}^+$
- **Private**: Each party P_b has private input $\mathsf{sk}_b = \lambda_b^{1-b}(0) \cdot f(b) \in \mathbb{Z}_q$

1. **Tag R from Threshold Signature:**
   (a) Run the first half of the threshold signing protocol
   $$(R, \mathsf{state}_b) \leftarrow \pi^{\mathsf{R}}_{\mathsf{Sign}}(\mathsf{sk}_b, 1-b, m)$$
2. **Sample New Polynomial:**
   (a) Send $(\texttt{sample-element}, \mathsf{id}^{\mathsf{coin}}, q)$ to $\mathcal{F}_{\mathsf{Coin}}$ and wait for response $(\mathsf{id}^{\mathsf{coin}}, \delta)$
   (b) Define degree-1 polynomial f_δ over $\mathbb{Z}_q$ such that
   $$f_\delta(0) = 0 \quad \text{and} \quad f_\delta(1) = \delta$$
   (c) Compute
   $$\mathsf{sk}'_b = \mathsf{sk}_b + f_\delta(b)$$
3. **Store Tagged Refresh:**
   (a) Retrieve Epoch index epoch
   (b) Append $(R, \mathsf{sk}'_b, \mathsf{epoch})$ to rpool
4. Complete the threshold signature protocol by running $\sigma \leftarrow \pi^{\sigma}_{\mathsf{Sign}}$
5. If $\sigma \neq \perp$ then set $\mathsf{tx} = (m, R, \sigma)$ and send $(\texttt{Submit}, sid, \mathsf{tx})$ to $\mathcal{G}_{\mathsf{Ledger}}$

Note that in Step 5 it is sufficient for only one party to send the transaction tx to the ledger.

While the above protocol generates candidate refresh polynomials, choosing which one to use from rpool (and when to delete old shares) is done separately. The idea is that when a new block is obtained from $\mathcal{G}_{\mathsf{Ledger}}$ the parties each scan it to find signatures under their shared public key pk. The signatures are cross-referenced with rpool tuples stored in memory by matching R (no two signatures will have the same R) and the ones without corresponding tuples are ignored. If any such signatures are found, the one occurring first in the block is chosen to signal the next refresh; in particular the corresponding sk'_b overwrites sk_b stored in

memory, rpool is erased, and the epoch counter is incremented.

Protocol 6.4.2. $\pi^{(2,2)}_{\rho\text{-update}}$**. Updating** $(2,2)$ **Threshold Schnorr State.**

Parameters: Elliptic Curve Group $(\mathbb{G}, G, q)$
Parties: P_i (local refresh protocol)
Ideal Oracles: $\mathcal{G}_{\mathsf{Ledger}}$
Inputs: Epoch counter epoch, a list $\mathsf{rpool} = \{(\mathsf{epoch}, \mathsf{sk}'_i, R)\}$, private key share sk_i.

1. Send (Read) to $\mathcal{G}_{\mathsf{Ledger}}$ and receive $(\mathtt{Read}, b)$. Set BLK to be the latest block occurring in b
2. Search for the first signature (σ, R) occurring in BLK under pk such that $\exists(R, \mathsf{sk}'_i, \mathsf{epoch}) \in \mathsf{rpool}$
3. Overwrite $\mathsf{sk}_i = \mathsf{sk}'_i$ and erase rpool
4. Set $\mathsf{epoch} = \mathsf{epoch} + 1$

It is clear that this protocol achieves all desired properties when both parties are honest. We give a proof of the extended $(2, n)$ protocol directly in the next section. However we make a few observations at this point that will aid in building the proof for the extended protocol.

Before and after a refresh the view of an adversary corrupting P_b when $\mathsf{epoch} = x$ is completely independent of the view when corrupting P_{1-b} after $\mathsf{epoch} = x + 1$. This is clear as polynomials f and f' are independently distributed, and so $\mathsf{sk}_b = f(b)$ can not be meaningfully combined with $\mathsf{sk}'_{1-b} = f'(1-b)$.

No two entries in rpool **will have the same** R by virtue of each R being chosen uniformly for each entry, the likelihood of there being two entries with the same R value in rpool is negligible, with about $\sqrt{q}$ signatures having to be generated before a collision occurs.

6.5 $(2, n)$ Refresh With Two Online

In this section, we give the malicious secure protocol for two online parties to coordinate an authenticated refresh of the secret key for arbitrarily many offline parties. We now describe how to ensure that offline parties can get up to speed upon waking up, crucially in a way that every party is in agreement about which polynomial to use so that sk_i erasures are always safe.

Goal Observe that if every party is in agreement about rpool, then the rest of the refresh procedure is deterministic and straightforward. Therefore it suffices to construct a mechanism to ensure that for each $(R, \mathsf{sk}'_b, \mathsf{epoch})$ tuple an online party P_b appends to its rpool, each offline party P_i is able to append a consistent value $(R, \mathsf{sk}'_i, \mathsf{epoch})$ to its own rpool. Here 'consistent' means that the points $(0, \mathsf{sk})$, (b, sk'_b), (i, sk'_i) are collinear.

An Attempt at a Solution We first note that since either one of the online parties P_b may be malicious and therefore unreliable, it simplifies matters to design the refresh protocol so that they both send the same message to an offline P_i. The message itself should deliver $f_\delta(i)$ (so that P_i can compute sk'_i) along with R. Simultaneously it must be ensured that a malicious party is unable to spoof such a message and confuse P_i.

In order to solve this problem, we take advantage of the fact that the parties already share a distributed key setup; as any two parties must be able to sign a message in a $(2, n)$ threshold signature scheme, we take advantage of this feature to authenticate sent messages with threshold signatures *internal* to the protocol. In particular, when any P_b, P_{1-b} agree on an entry (R, sk_b) to add to rpool, they also produce a threshold signature z under the shared public key pkauthenticating this entry. Each P_b is instructed to send the new rpool entry accompanied by its signature z to every offline party. If at least one of P_b, P_{1-b} follows the protocol (note that only one may be corrupt), every offline party will have received the new rpool entry when it wakes up. Additionally due to the same reason that $(2, n)$ signatures are unforgeable by an adversary corrupting a single party, such an adversary will be unable to convince any offline P_i to add an entry to rpool that was not approved by an honest party. An implication of this unforgeability feature is that an offline party can safely ignore received messages that are malformed.

A Subtle Attack Again the inherent unfairness of two-party computation stands in the way of achieving a consistent rpool. In particular an adversary corrupting P_b^* may choose to abort the computation the moment she receives the internal threshold signature z, denying the online honest party P_{1-b} this value and therefore removing its ability to convince its offline friends to add the new rpool entry. This is a dangerous situation, as P_b^* now has the power to control whether the offline parties update rpool or not, i.e. by choosing whether or not to send the new rpool entry (which it can convince offline parties to use as it has z). While this will not immediately constitute a breach of privacy, the fact that honest parties do not agree on rpool could violate unanimous erasure; at best this requires *all* honest parties to come online to re-share the secret, and at worst this could mean that the secret key is lost forever (e.g. in the (2,3) cold storage use case).

Our Solution This is where it is crucial that the first half of the threshold signing protocol ($\pi^{\mathsf{R}}_{\mathsf{Sign}}$) is simulatable without the signature σ itself; in fact it is the entire reason for this choice of abstraction. Assume that P_{1-b} updates its rpool with the new value before even producing z. Following this, P_{1-b} will refuse to instruct $\mathcal{F}^{n,2}_{\mathsf{Sign}}$ to reveal the signature σ until it is in possession of the local threshold signature z to send to offline parties. There are now two choices that P^*_b has when executing the attack described above:

- **Update rpool of offline parties**: i.e. the adversary chooses to add (R, f_δ) to the rpool of some/all offline parties. In this case, in order to actually exploit the inconsistency between rpool of different honest parties, the adversary must trigger a refresh that produces different outcomes for different rpool. Specifically, the signature σ under public key $\mathsf{pk}$and the nonce R must appear on the blockchain; i.e. the same R that P_b interrupted signing with P_{1-b} but sent to offline parties. However since protocol $\pi^{\mathsf{R}}_{\mathsf{Sign}}$ by itself keeps σ completely hidden and P_{1-b} does not continue with $\pi^{\sigma}_{\mathsf{Sign}}$, the task of the adversary is essentially to produce σ under a specific uniformly chosen R (of unknown discrete logarithm). We show that this amounts to solving the discrete logarithm problem in the curve $\mathbb{G}$.

- **Do not update rpool of offline parties**: All honest parties have the same rpool anyway, and there is no point of concern.

Therefore instead of using complicated mechanisms (eg. forcing everyone to come online, extra messages on the blockchain, etc.) to ensure that every honest party agrees on the same rpool, we design our protocol so that any inconsistencies in rpool are inconsequential.

We present the protocol below, which includes some optimizations and notation omitted from the above explanation.

Protocol 6.5.1. $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$**.** $(2, n)$ **Schnorr Signing With Refreshment**

Parameters: Elliptic Curve Group $(\mathbb{G}, G, q)$
Parties: P_b for $b \in [n]$
Ideal Oracles: $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$, $\mathcal{G}_{\mathsf{Ledger}}$, random oracle RO
Inputs:

- **Common**: Message to be signed $m \in \{0,1\}^*$, public key $\mathsf{pk} \in \mathbb{G}$, each party's share in the exponent $\mathsf{pk}_b = \lambda_b^{1-b}(0) \cdot F(b)$ where F is the polynomial over $\mathbb{G}$ passing through $(0, \mathsf{pk})$ and $(b, f(b) \cdot G)$, epoch index $\mathsf{epoch} \in \mathbb{Z}^+$
- **Private**: Each party P_b has private input $\mathsf{sk}_b = \lambda_b^{1-b}(0) \cdot f(b) \in \mathbb{Z}_q$

---

1. **Tag R from Threshold Signature:** (*identical to* $\pi^{(2,2)}_{\rho\text{-}\mathsf{sign}}$)

2. **Sample New Polynomial:** (*identical to* $\pi^{(2,2)}_{\rho\text{-sign}}$)

3. **Store Tagged Refresh:**

   (a) Append $(R, \mathsf{sk}'_b, \mathsf{epoch})$ to rpool

   (b) Establish common nonce $K \in \mathbb{G}$ along with an additive sharing of its discrete logarithm:

      i. Sample $k_b \leftarrow \mathbb{Z}_q$, set $K_b = k_b \cdot G$ and send $(\texttt{com-proof}, \mathsf{id}_b^{\mathsf{com\text{-}zk}}, k_b, K_b)$ to $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$

      ii. Upon receiving $(\texttt{committed}, 1-b, \mathsf{id}_{1-b}^{\mathsf{com\text{-}zk}})$ from $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$, send $(\texttt{open}, \mathsf{id}_b^{\mathsf{com\text{-}zk}})$ to $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$

      iii. Wait to receive $(\texttt{decommitted}, 1-b, \mathsf{id}_{1-b}^{\mathsf{com\text{-}zk}}, K_{1-b} \in \mathbb{G})$ from $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$

      iv. Set $K = K_b + K_{1-b}$

   (c) Compute

$$e = \mathsf{RO}(R||K||\delta||\mathsf{epoch})$$
$$z_b = e \cdot \mathsf{sk}_b + k_b$$

   (d) Send z_b to P_{1-b} and wait for z_{1-b}, upon receipt verifying that

$$z_{1-b} \cdot G = e \cdot \mathsf{pk}_{1-b} + K_{1-b}$$

   and compute $z = z_b + z_{1-b}$

   (e) Set $msg = (R, \mathsf{epoch}, \delta, K, z)$

   (f) For each $i \in [n] \setminus \{b, 1-b\}$, send msg to P_i

4. Complete the threshold signature protocol by running $\sigma \leftarrow \pi^{\sigma}_{\mathsf{Sign}}$

5. If $\sigma \neq \perp$ then set $\mathsf{tx} = (m, R, \sigma)$ and send $(\texttt{Submit}, sid, \mathsf{tx})$ to $\mathcal{G}_{\mathsf{Ledger}}$

We now specify the refresh procedure for a party P_i to process its received messages, reconstruct rpool, and shift to the latest shared polynomial. This refresh procedure is general so that parties who were offline for a number of epochs can catch up.

Protocol 6.5.2. $\pi^{(2,n)}_{\rho\text{-update}}$**. Applying Update Packages to** $(2, n)$ **Schnorr State**

Parameters: Elliptic Curve Group $(\mathbb{G}, G, q)$
Parties: P_i (local refresh protocol)
Ideal Oracles: $\mathcal{G}_{\mathsf{Ledger}}$
Inputs: Epoch counter epoch, a list $\mathsf{rpool} = \{(\mathsf{epoch}, \mathsf{sk}'_i, R)\}$, public key pk, private key share sk_i (define $\mathsf{pk}_i = \mathsf{sk}_i \cdot G$).

1. For each unique msg received when offline do the following:
   (a) Parse $(R, \mathsf{epoch}', \delta, K, z) \leftarrow msg$ and if $\mathsf{epoch}' < \mathsf{epoch}$ ignore this msg
   (b) Compute $e = \mathsf{RO}(R||K||\delta||\mathsf{epoch}')$ and verify that
   $$z \cdot G = e \cdot \mathsf{pk} + K$$
   (c) Define degree-1 polynomial f_δ over $\mathbb{Z}_q$ such that
   $$f_\delta(0) = 0 \quad \text{and} \quad f_\delta(1) = \delta$$
   and interpolate $\delta_i = f_\delta(i)$
   (d) If $\mathsf{epoch}' = \mathsf{epoch}$, compute
   $$\mathsf{sk}_i' = \mathsf{sk}_i + \delta_i$$
   and append $(R, \mathsf{sk}_i', \mathsf{epoch})$ to rpool
   (e) Otherwise $\mathsf{epoch}' > \mathsf{epoch}$ so append $(\mathsf{epoch}', \delta_i, R)$ to fpool
2. Send $(\mathtt{Read})$ to $\mathcal{G}_{\mathsf{Ledger}}$ and receive $(\mathtt{Read}, b)$ in response. Set BLK to be the latest blocks occurring in b since last awake, and in sequence from the earliest block, for each (σ, R) under pk encountered do the following:
   (a) Find $(R, \mathsf{sk}_i', \mathsf{epoch}) \in \mathsf{rpool}$ (match by R), ignore σ if not found
   (b) Overwrite $\mathsf{sk}_i = \mathsf{sk}_i'$, set $\mathsf{epoch} = \mathsf{epoch} + 1$, and set $\mathsf{rpool} = \varnothing$
   (c) For each $(\mathsf{epoch}, \delta_i, R) \in \mathsf{fpool}$ (i.e. matching current epoch) do:
      i. Set $\mathsf{sk}_i' = \mathsf{sk}_i + \delta_i$
      ii. Append $(R, \mathsf{sk}_i', \mathsf{epoch})$ to rpool
      iii. Remove this entry from fpool

In the above refresh protocol $\pi^{(2,n)}_{\rho\text{-}\mathsf{update}}$, the set rpool will always be consistent across honest parties (except for inconsequential differences) and fpool will be empty by the end. This is due to the fact that fpool contains candidate refresh values intended for epoch values further than the one "caught up with" so far; no honest party will approve a candidate with a higher epoch counter than its own, and every honest party reaches the same epoch value upon refresh. Further details can be found in the section addressing non-degeneracy of the protocol in the proof that follows.

Theorem 6.5.3. *If* $(\pi^{\mathsf{DKG}}_{\mathsf{Setup}}, \pi^{\mathsf{R}}_{\mathsf{Sign}}, \pi^{\sigma}_{\mathsf{Sign}})$ *is a threshold signature scheme for signing equation* Sign*, and the discrete logarithm problem is hard in* $\mathbb{G}$*, then* $(\pi^{\mathsf{DKG}}_{\mathsf{Setup}}, \pi^{(2,n)}_{\rho\text{-}\mathsf{update}})$ *UC-realizes* $\mathcal{F}^{n,2}_{\mathsf{Sign}}$

in the $(\mathcal{G}_{\mathsf{Ledger}}, \mathcal{F}_{\mathsf{Com-ZK}}^{\mathsf{R}_{DL}})$*-hybrid model in the presence of a mobile adversary corrupting one party, with offline refresh.*

Proof. (Sketch) The protocol $\pi_{\mathsf{Setup}}^{\mathsf{DKG}}$ can be simulated the standard way, with the corrupt party P_i's key share sk_i remembered as output. We now describe the simulator $\mathcal{S}_{\rho\text{-}\mathsf{sign}}^{(2,n)}$ for protocol $\pi_{\rho\text{-}\mathsf{sign}}^{(2,n)}$. This simulator is given sk_i as input, and outputs (R, sk_i').

Simulator 6.5.4. $\mathcal{S}_{\rho\text{-}\mathsf{sign}}^{(2,n)}$

Parameters: Elliptic Curve Group $(\mathbb{G}, G, q)$
Ideal Oracles Controlled: $\mathcal{F}_{\mathsf{Com-ZK}}^{\mathsf{R}_{DL}}$, random oracle RO
Ideal Oracles Not Controlled: $\mathcal{G}_{\mathsf{Ledger}}$
Inputs:

- **Common**: Message to be signed $m \in \{0,1\}^*$, public key $\mathsf{pk} \in \mathbb{G}$, each party's share in the exponent $F(b) = f(b) \cdot G$, epoch index $\mathsf{epoch} \in \mathbb{Z}^+$
- **Private**: P_b's key share $\mathsf{sk}_b = f(b) \in \mathbb{Z}_q$

---

1. **Tag R from Threshold Signature:**

   (a) Simulate the first half of the threshold signing protocol

   $$(R, \mathsf{state}_b) \leftarrow \mathcal{S}_{\mathsf{Sign}}^{\mathsf{R}}(\mathsf{sk}_b, 1-b, m)$$

   relaying $(\texttt{get-instance-key}, \mathsf{id}^{\mathsf{sig}})$ and $(\texttt{instance-key}, \mathsf{id}^{\mathsf{sig}}, R)$ between $\mathcal{S}_{\mathsf{Sign}}^{\mathsf{R}}$ and $\mathcal{F}_{\mathsf{Sign}}^{n,2}$ when required.

2. **Sample New Polynomial:** (*identical to* $\pi_{\rho\text{-}\mathsf{sign}}^{(2,2)}$)

   (a) Sample $\delta \leftarrow \mathbb{Z}_q$ and send $(\mathsf{id}^{\mathsf{coin}}, \delta)$ to P_b on behalf of $\mathcal{F}_{\mathsf{Coin}}$

   (b) Define degree-1 polynomial f_δ over $\mathbb{Z}_q$ such that

   $$f_\delta(0) = 0 \quad \text{and} \quad f_\delta(1) = \delta$$

   (c) Compute

   $$\mathsf{sk}_b' = \mathsf{sk}_b + f_\delta(b)$$

3. **Store Tagged Refresh:**

   (a) Simulate a signature $R, \delta, \mathsf{epoch}$ under pk_{1-b}:

i. Sample $z_{1-b} \leftarrow \mathbb{Z}_q$ and $e \leftarrow \mathbb{Z}_q$ uniformly at random

ii. Compute

$$K = z \cdot G - e \cdot \mathsf{pk}_{1-b}$$

iii. Program $\mathsf{RO}(R||K||\delta||\mathsf{epoch}) = e$

(b) Establish common nonce $K \in \mathbb{G}$:

i. Send $(\texttt{committed}, 1-b, \mathsf{id}^{\mathsf{com\text{-}zk}}_{1-b})$ to P^*_b on behalf of $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$

ii. Receive $(\texttt{com-proof}, \mathsf{id}^{\mathsf{com\text{-}zk}}_b, k_b, K_b)$ on behalf of $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$

iii. Set $K_{1-b} = K - K_b$

iv. Send $(\texttt{decommitted}, 1-b, \mathsf{id}^{\mathsf{com\text{-}zk}}_{1-b}, K_{1-b} \in \mathbb{G})$ to P^*_b on behalf of $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$

v. Wait for $(\texttt{open}, \mathsf{id}^{\mathsf{com\text{-}zk}})$ from P_b, upon receipt sending z_{1-b} in response

(c) Wait for z_b, upon receipt verifying that

$$z_b = e \cdot \mathsf{sk}_b + k_b$$

4. Simulate the rest of the threshold signature protocol by running $\mathcal{S}^{\sigma}_{\mathsf{Sign}}(\mathsf{state}_b)$ relaying $(\texttt{proceed}, \mathsf{id}^{\mathsf{sig}})$ and $(\texttt{signature}, \mathsf{id}^{\mathsf{sig}}, \sigma)$ between P^*_b and $\mathcal{F}^{n,2}_{\mathsf{Sign}}$ as necessary.

5. If P^*_b asks $\mathcal{F}^{n,2}_{\mathsf{Sign}}$ to release σ to P_{1-b}, then set $\mathsf{tx} = (m, R, \sigma)$ and send $(\texttt{Submit}, sid, \mathsf{tx})$ to $\mathcal{G}_{\mathsf{Ledger}}$

6. Output (R, sk'_b)

Simulating $\pi^{(2,n)}_{\rho\text{-}\mathsf{update}}$is simple: every time the adversary $\mathcal{Z}$ sends a $(\texttt{sign}, m, i, j)$ command to a pair of honest parties, the simulator obtains a signature R, σ from $\mathcal{F}^{n,2}_{\mathsf{Sign}}$, samples $\delta \leftarrow \mathbb{Z}_q$, and simulates a local signature z under $\mathsf{pk}$to authenticate $R, \delta, \mathsf{epoch}$ just as in Step 3a of Simulator $\mathcal{S}^{(2,n)}_{\rho\text{-}\mathsf{sign}}$above. It sets $msg = (R, \mathsf{epoch}, \delta, K, z)$ and makes msg available to the corrupt party.

We now sketch an argument that the distribution of the real protocol is computationally indistinguishable from the ideal one.

We can progressively substitute each instance of $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$run with honest parties belonging to an epoch with $\mathcal{S}^{(2,n)}_{\rho\text{-}\mathsf{sign}}$run with $\mathcal{F}^{n,2}_{\mathsf{Sign}}$. The distinguishing advantage of $\mathcal{Z}$ at each step is bounded by the advantage of a PPT adversary distinguishing $(\pi^{\mathsf{DKG}}_{\mathsf{Setup}}, \pi^{\mathsf{R}}_{\mathsf{Sign}}, \pi^{\sigma}_{\mathsf{Sign}})$ from the corresponding ideal executions with $\mathcal{F}^{n,2}_{\mathsf{Sign}}$ as produced by simulators $(\mathcal{S}^{\mathsf{DKG}}_{\mathsf{Setup}}, \mathcal{S}^{\mathsf{R}}_{\mathsf{Sign}}, \mathcal{S}^{\sigma}_{\mathsf{Sign}})$, which is assumed to be negligible. In order to extend this strategy to a mobile adversary, it suffices to argue that the polynomials f, f' used to share skappear independently distributed before and after a refresh. This follows immediately from the fact that an adversary who jumps

from party P_i to P_j is given $f(i)$ and $f'(j)$ but does not see the difference f_δ between f, f', just as discussed in the (2,2) case in Section 6.4.

It remains to be argued that the protocol is not degenerate. The non-degeneracy property is achieved by fulfilling two important requirements:

System Epoch Increments When the parties executing $\pi^{(2,n)}_{\rho\text{-sign}}$ are honest, the system epoch will always increment upon the next refresh command, i.e. if $\pi^{(2,n)}_{\rho\text{-sign}}$ is run by honest parties with counter epoch, then every subsequent execution of $\pi^{(2,n)}_{\rho\text{-update}}$ by any party in the system will result in a local epoch counter of at least $\mathsf{epoch} + 1$. This is easy to see for this protocol, as honest parties executing $\pi^{(2,n)}_{\rho\text{-sign}}$ will always produce a signature σ which will subsequently appear on the blockchain (after delay T as per $\mathcal{G}_{\mathsf{Ledger}}$). Simultaneously every party will find a corresponding update to rpool sent to it, which will be applied by $\pi^{(2,n)}_{\rho\text{-update}}$ when σ appears on the blockchain.

Consistency Every honest party outputs the same epoch counter upon executing $\pi^{(2,n)}_{\rho\text{-update}}$ simultaneously. As alluded to earlier in Section 6.5 proving this amounts to showing that the state of rpool maintained by each honest party differs inconsequentially. In particular, let P_i and P_j be honest parties maintaining rpool_i and rpool_j respectively such that $\exists(R, \mathsf{sk}'_i, \mathsf{epoch}) \in \mathsf{rpool}_i$ but $\nexists(R, \mathsf{sk}'_j, \mathsf{epoch}) \in \mathsf{rpool}_j$. First we claim that $(R, \mathsf{sk}'_i, \mathsf{epoch})$ can be traced to a unique execution of $\pi^{(2,n)}_{\rho\text{-sign}}$ between a corrupt party P^*_b and honest party P_{1-b}. There are only two alternative events: (1) that there is a collision in R values generated by two protocol instances (occurs with probability $|\mathbf{m}|^2/2q$ where $|\mathbf{m}|$ is the number of messages signed), or (2) P_i received z authenticating this entry without any honest party's help in its creation; the exact same technique to prove (threshold) Schnorr signatures secure can be employed here to construct a reduction to the Discrete Logarithm problem in curve $\mathbb{G}$ (if this event occurs with probability ϵ then there is a reduction to DLog successful with probability $\epsilon/|\mathbf{m}|$). Given that $(R, \mathsf{sk}'_i, \mathsf{epoch})$ can be traced to a unique execution of $\pi^{(2,n)}_{\rho\text{-sign}}$ between P^*_b and P_{1-b} it must be the case that P^*_b aborted the comptation at Step 3d, i.e. P^*_b received z to authenticate this entry but withheld this value from P_{1-b} (or else P_j would have received this entry when offline as well due to P_{1-b}). Observe that this inconsistency in $\mathsf{rpool}_i, \mathsf{rpool}_j$ is consequential only if (σ, R) appears on $\mathcal{G}_{\mathsf{Ledger}}$, despite the fact that P_{1-b} will not execute $\pi^{\sigma}_{\mathsf{Sign}}$ to produce this value. We show that if this event happens with probability ϵ then there is an adversary for the DLog problem successful with probability $\epsilon/|\mathbf{m}|$. This is because R is chosen uniformly in $\pi^{(2,n)}_{\rho\text{-sign}}$ (ie. internally by $\pi^{\mathsf{R}}_{\mathsf{Sign}}$ as it realizes $\mathcal{F}^{n,2}_{\mathsf{Sign}}$) and the task of $\mathcal{Z}$ is to produce σ that verifies under uniformly chosen nonce R and public key pk. We can use such a $\mathcal{Z}$ to solve the DLog problem in $\mathbb{G}$ as follows:

1. Receive $X \in \mathbb{G}$ from the DLog challenger.
2. Choose $\mathsf{sk} \leftarrow \mathbb{Z}_q$, set $\mathsf{pk} = \mathsf{sk} \cdot G$
3. Run $\mathcal{S}^{\mathsf{DKG}}_{\mathsf{Setup}}$ for $\mathcal{Z}$ with pkprogrammed to be the public key.
4. For each message $m \in \mathbf{m}$ except one, run $\mathcal{S}^{(2,n)}_{\rho\text{-}\mathsf{sign}}$ as required to simulate $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$while also acting on behalf of $\mathcal{F}^{n,2}_{\mathsf{Sign}}$
5. For one randomly chosen instance of $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$, use $\mathcal{S}^{\mathsf{R}}_{\mathsf{Sign}}$ to program X as the signing nonce R.
6. If the correct instance of $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$is chosen, P^*_b will abort this protocol before the corresponding σ has to be released, and yet σ still appears on $\mathcal{G}_{\mathsf{Ledger}}$
7. If σ is obtained from $\mathcal{G}_{\mathsf{Ledger}}$, solve for x such that $x \cdot G = X$ as a function of σ, sk as per the signing equation Sign. This is dependent on the equation Sign itself, but it is straightforward how to retrieve the instance key x given the secret key sk and signature σ as per $\mathsf{Sign}_{\text{ECDSA}}$ and $\mathsf{Sign}_{\text{ECDSA}}$.

The above reduction succeeds when $\mathcal{Z}$ induces this event (probability ϵ) and the correct instance of $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$is chosen (probability $1/|\mathbf{m}|$) bringing the total success probability to $\epsilon/|\mathbf{m}|$.

As the simulated distribution is indistinguishable from the execution of the real protocol and the protocol is non-degenerate, this proves the theorem.

□

An Optimization We note that one can save a query to $\mathcal{F}_{\mathsf{Coin}}$ and a $\mathbb{Z}_q$ element from being having to be sent by defining $\delta = \mathsf{RO}(R||K||\mathsf{epoch})$ instead of computing it separately from the internal threshold signature z. As (R, K) guarantee κ bits of entropy, the resulting δ will be distributed uniformly.

6.6 Proactive $(2, n)$ ECDSA

Computing $(2, n)$ ECDSA signatures is significantly more difficult than Schnorr, due to the non-linear nature of the ECDSA signing equation. As a result, all such recent threshold ECDSA protocols [GG18, LNR18, DKLs18, DKLs19] make use of a secure multiplication functionality (or equivalent protocol) $\mathcal{F}_{\mathsf{MUL}}$ in their signing phases. If $\mathcal{F}_{\mathsf{MUL}}$ were to be instantiated independently for each threshold ECDSA signature produced, we could just use the same strategy as in the previous section, since the $\pi^{\mathsf{R}}_{\mathsf{Sign}}$ protocol would take only key

shares as arguments. However $\mathcal{F}_{\mathsf{MUL}}$ is expensive to realize for individual invocations, and given that threshold signature protocols already need a "preprocessing" phase for key generation (ie. $\pi^{\mathsf{DKG}}_{\mathsf{Setup}}$), all the cited works make use of this phase to also run some preprocessing for $\mathcal{F}_{\mathsf{MUL}}$ to make its invocation during signing cheaper. Therefore, we also need to change how we deal with proactively refreshing the shares. In a nutshell, the main technical challenge we address in this section is that now the parties, on top of their key shares, also include in their persistent storage some state information for the $\mathcal{F}_{\mathsf{MUL}}$ protocol and that this state is a new target for a mobile adversary. Therefore, the state needs to be refreshed as well.

We start by abstracting the two-party multiplication protocol ($\pi^{\mathsf{Setup}}_{\mathsf{MUL}}$,$\pi^{\mathsf{Online}}_{\mathsf{MUL}}$) used within ECDSA threshold protocols. The protocols are run by party P_i with P_j as the counterparty as follows,

- $(\mathsf{state}^{i,j}_{\mathsf{MUL}} \in \{0,1\}^*) \leftarrow \pi^{\mathsf{Setup}}_{\mathsf{MUL}}(j)$
- $(t_i \in \mathbb{Z}_q) \leftarrow \pi^{\mathsf{Online}}_{\mathsf{MUL}}\left(\mathsf{state}^{i,j}_{\mathsf{MUL}}, x_j\right)$

The pair of protocols ($\pi^{\mathsf{Setup}}_{\mathsf{MUL}}$,$\pi^{\mathsf{Online}}_{\mathsf{MUL}}$) must realize $\mathcal{F}_{\mathsf{MUL}}$. As per the functionality specification, $t_i + t_j = x_i \cdot x_j$ after $\pi^{\mathsf{Online}}_{\mathsf{MUL}}$ is run, and this can be done arbitrarily many times for different inputs. Every pair of parties in the system shares an instantiation of $\mathcal{F}_{\mathsf{MUL}}$, and so P_i maintains $\mathsf{state}^{i,j}_{\mathsf{MUL}}$ for each $j \in [n] \setminus i$. Therefore in our abstraction for threshold ECDSA protocols ($\pi^{\mathsf{DKG}}_{\mathsf{Setup}}, \pi^{\mathsf{Setup}}_{\mathsf{MUL}}, \pi^{\mathsf{R}}_{\mathrm{ECDSA}}, \pi^{\sigma}_{\mathrm{ECDSA}}$) we include the state required by P_i for multiplication with P_j as an argument for online signing. We avoid rewriting the formal abstraction for readability, as it is essentially a reproduction of Section 6.3 with the inclusion of $\mathsf{state}^{i,j}_{\mathsf{MUL}}$ as an argument/output in the correct places.

The same restrictions on the simulators for these protocols hold, see Section 6.3 for details. It is not hard to show that the recent protocols of Lindell et al. [LNR18], Gennaro and Goldfeder [GG18], and Doerner et al. [DKLs19] fit these characterizations. The inclusion of $\{\mathsf{state}^{i,j}_{\mathsf{MUL}}\}_{j\in[n]}$ as persistent state that parties must maintain across signatures creates an additional target that must be defended from a mobile adversary. We show how here to refresh $\{\mathsf{state}^{i,j}_{\mathsf{MUL}}\}_{j\in[n]}$ required by the OT-based instantiation of $\mathcal{F}_{\mathsf{MUL}}$ (as in Doerner et al. [DKLs19]) and consequently upgrade compatible threshold ECDSA protocols [DKLs19, GG18, LNR18] to proactive security.

Approach The setup used by the multiplier of Doerner et al. consists of a number of base OTs which are "extended" for use online [KOS15]. These base OTs are the only component of their multiplier which requires each party to keep private state. Therefore re-randomizing these OTs in the interval between an adversary's jump from one party to the other is sufficient to maintain security. The central idea to implement this re-randomization is to apply the

approach introduced by Beaver [Bea95] of "adjusting" preprocessed OTs once inputs are known online.

6.6.1 Proactive Secure Multiplication

We begin by describing how two parties can re-randomize OT itself, and then describe how to apply this technique to re-randomize OT Extensions.

Re-randomizing Oblivious Transfer Assume that Alice has two uniform κ-bit strings r_0, r_1, and Bob has a bit b and correspondingly the string r_b. Let $rand \leftarrow \{0,1\}^{2\kappa+1}$ be a uniformly chosen string that is parsed into chunks $r'_0, r'_1 \in \{0,1\}^\kappa$ and $b' \in \{0,1\}$ by both parties. The re-randomization process for Alice ($\mathsf{Refresh\_OT_A}$) and Bob ($\mathsf{Refresh\_OT_B}$) is non-interactive (given $rand$) and proceeds as follows:

1. $\mathsf{Refresh\_OT_A}\left((r_0, r_1), rand\right)$: output $r''_0 = r_{b'} \oplus r'_0$ and $r''_1 = r_{1-b'} \oplus r'_1$
2. $\mathsf{Refresh\_OT_B}\left((b, r_b), rand\right)$: output $b'' = b \oplus b'$ and $r''_{b''} = r_b \oplus r'_{b''}$
3. Alice now holds (r''_0, r''_1) and Bob holds $b'', r''_{b''}$

It is clear to see that Alice and Bob learn nothing of each other's private values, only the offsets r'_0, r'_1, b' between the new and old ones. Consider the view of a mobile adversary that jumps from one party to the other.

- Alice $\rightarrow$ Bob: (r_0, r_1) before the refresh, and $(b'', r''_{b''})$ after the refresh.
- Bob $\rightarrow$ Alice: (b, r_b) before the refresh, and (r''_0, r''_1) after the refresh.

Assuming that r'_0, r'_1, b' are hidden and that these values are uniformly chosen, in both the above cases the adversary's view before and after the refresh are completely independent.

Re-randomizing OT Extensions The persistent state maintained by OT Extension protocols based on that of Ishai et al. [IKNP03] consists of the result of a number of OTs performed during a preprocessing phase. Re-randomizing this state can be done by simply repeating the above protocol for each preprocessed OT instance. Indeed, the instantiation of OT Extension implemented by Doerner et al. is the protocol of Keller et al. [KOS15] which is captured by this framework.

Re-randomizing multipliers There is no further persistent state maintained across $\mathcal{F}_{\mathsf{MUL}}$ invocations by the protocol of Doerner et al. [DKLs19], and so we leave implicit the construction of $\mathsf{state}_{\mathsf{MUL}}' \leftarrow \mathsf{Refresh\_MUL}(\mathsf{state}_{\mathsf{MUL}}, rand)$. The only missing piece is how $rand$ is chosen; in the context of the multipliers in isolation, this value can be thought of coming from a coin-tossing protocol that is invisible to the adversary (when neither party is corrupt).

6.6.2 Multiplier Refresh in $(2, n)$ ECDSA

The previous subsection describes how to realize $\mathcal{F}_{\mathsf{MUL}}$ with proactive security when a mechanism to agree on when/which $rand$ to use is available. Fortunately the protocol described in Section 6.5 provides exactly such a mechanism for the $(2, n)$ threshold signature setting. We briefly describe how to augment Protocol 6.5.1 to produce the randomness $rand$ required to proactivize multipliers in addition to the distributed key shares.

$(2, n)$ **Offline Refresh** The two online parties P_b, P_{1-b} engage in a coin-tossing protocol in the **Sample New Polynomial** phase to produce a uniform κ-bit value seed. In the **Store Tagged Refresh** phase they include seed to be stored in rpool along with corresponding $\mathsf{epoch}, \mathsf{sk}_b', R$ (and communicate seed to offline parties along with these values). If the signature using R is used to signal a refresh, then seed is expanded by every pair of parties to produce $rand$ as necessary.

We give the entire protocol below for completeness. We give the full proactive ECDSA protocol below. It shares many similarities with $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$and so we underline changes in this protocol.

Protocol 6.6.1. $\pi^{(2,n)}_{\rho\text{-}\mathbf{ECDSA}}$

Parameters: Elliptic Curve Group $(\mathbb{G}, G, q)$

Parties: P_b for $b \in [n]$

Ideal Oracles: $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$, $\mathcal{G}_{\mathsf{Ledger}}$, random oracle RO

Inputs:

- **Common**: Message to be signed $m \in \{0,1\}^*$, public key $\mathsf{pk} \in \mathbb{G}$, each party's share in the exponent $\mathsf{pk}_b = \lambda_b^{1-b}(0) \cdot F(b)$ where F is the polynomial over $\mathbb{G}$ passing through $(0, \mathsf{pk})$ and $(b, f(b) \cdot G)$, epoch index $\mathsf{epoch} \in \mathbb{Z}^+$
- **Private**: Each party P_b has private input $\mathsf{sk}_b = \lambda_b^{1-b}(0) \cdot f(b) \in \mathbb{Z}_q$

---

1. **Tag R from Threshold Signature:**

(a) Run the first half of the threshold signing protocol

$$(R, \mathsf{state}_b) \leftarrow \pi^{\mathsf{R}}_{\mathsf{Sign}}\left(\mathsf{sk}_b, 1-b, \underline{\mathsf{state}^{b,1-b}_{\mathsf{MUL}}}, m\right)$$

2. **Sample New Polynomial:**

(a) Send $(\texttt{sample-element}, \mathsf{id}^{\mathsf{coin}}_1, q)$ and $\underline{(\texttt{sample-element}, \mathsf{id}^{\mathsf{coin}}_2, q)}$ to $\mathcal{F}_{\mathsf{Coin}}$ and wait for responses $(\mathsf{id}^{\mathsf{coin}}_1, \delta)$ and $\underline{(\mathsf{id}^{\mathsf{coin}}_2, \mathsf{seed})}$ respectively

(b) Define degree-1 polynomial f_δ over $\mathbb{Z}_q$ such that

$$f_\delta(0) = 0 \quad \text{and} \quad f_\delta(1) = \delta$$

(c) Compute

$$\mathsf{sk}'_b = \mathsf{sk}_b + f_\delta(b)$$

3. **Store Tagged Refresh:**

(a) Append $(R, \mathsf{sk}'_b, \underline{\mathsf{seed}}, \mathsf{epoch})$ to rpool

(b) Establish common nonce $K \in \mathbb{G}$ along with an additive sharing of its discrete logarithm:

i. Sample $k_b \leftarrow \mathbb{Z}_q$, set $K_b = k_b \cdot G$ and send $(\texttt{com-proof}, \mathsf{id}^{\mathsf{com\text{-}zk}}_b, k_b, K_b)$ to $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$

ii. Upon receiving $(\texttt{committed}, 1-b, \mathsf{id}^{\mathsf{com\text{-}zk}}_{1-b})$ from $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$, send $(\texttt{open}, \mathsf{id}^{\mathsf{com\text{-}zk}}_b)$ to $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$

iii. Wait to receive $(\texttt{decommitted}, 1-b, \mathsf{id}^{\mathsf{com\text{-}zk}}_{1-b}, K_{1-b} \in \mathbb{G})$ from $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$

iv. Set $K = K_b + K_{1-b}$

(c) Compute

$$e = \mathsf{RO}(R||K||\underline{\mathsf{seed}}||\delta||\mathsf{epoch})$$
$$z_b = e \cdot \mathsf{sk}_b + k_b$$

(d) Send z_b to P_{1-b} and wait for z_{1-b}, upon receipt verifying that

$$z_{1-b} \cdot G = e \cdot \mathsf{pk}_{1-b} + K_{1-b}$$

and compute $z = z_b + z_{1-b}$

(e) Set $msg = (R, \mathsf{epoch}, \delta, \underline{\mathsf{seed}}, K, z)$

(f) For each $i \in [n] \setminus \{b, 1-b\}$, send msg to P_i

4. Complete the threshold signature protocol by running $\sigma \leftarrow \pi^{\sigma}_{\mathsf{Sign}}$

5. If $\sigma \neq \perp$ then set $\mathsf{tx} = (m, R, \sigma)$ and send $(\texttt{Submit}, sid, \mathsf{tx})$ to $\mathcal{G}_{\mathsf{Ledger}}$

Update:

1. For each unique msg received when offline do the following:

   (a) Parse $(R, \mathsf{epoch}', \delta, \underline{\mathsf{seed}}, K, z) \leftarrow msg$ and if $\mathsf{epoch}' < \mathsf{epoch}$ ignore this msg

   (b) Compute $e = \mathsf{RO}(R||K||\underline{\mathsf{seed}}||\delta||\mathsf{epoch}')$ and verify that

   $$z \cdot G = e \cdot \mathsf{pk} + K$$

   (c) Define degree-1 polynomial f_δ over $\mathbb{Z}_q$ such that

   $$f_\delta(0) = 0 \quad \text{and} \quad f_\delta(1) = \delta$$

   and interpolate $\delta_i = f_\delta(i)$

   (d) If $\mathsf{epoch}' = \mathsf{epoch}$, compute

   $$\mathsf{sk}'_i = \mathsf{sk}_i + \delta_i$$

   and append $(R, \mathsf{sk}'_i, \underline{\mathsf{seed}}, \mathsf{epoch})$ to rpool

   (e) Otherwise $\mathsf{epoch}' > \mathsf{epoch}$ so append $(\mathsf{epoch}', \delta_i, \mathsf{seed}, R)$ to fpool

2. Send $(\texttt{Read})$ to $\mathcal{G}_{\mathsf{Ledger}}$ and receive $(\texttt{Read}, b)$ in response. Set BLK to be the latest blocks occurring in b since last awake, and in sequence from the earliest block, for each (σ, R) under pk encountered do the following:

   (a) Find $(R, \mathsf{sk}'_i, \underline{\mathsf{seed}}, \mathsf{epoch}) \in \mathsf{rpool}$ (match by R), ignore σ if not found

   (b) Overwrite $\mathsf{sk}_i = \mathsf{sk}'_i$, set $\mathsf{epoch} = \mathsf{epoch} + 1$, and set $\mathsf{rpool} = \varnothing$

   (c) <u>For each $j \in [n] \setminus i$ compute</u>

   $$\underline{rand_{ij} = \mathsf{RO}(i, j, \mathsf{seed})}$$

   <u>and overwrite</u>

   $$\underline{\mathsf{state}_{\mathsf{MUL}ij} = \mathsf{Refresh\_MUL}(\mathsf{state}_{\mathsf{MUL}ij}, rand_{ij})}$$

(d) For each $(\mathsf{epoch}, \delta_i, \underline{\mathsf{seed}}, R) \in \mathsf{fpool}$ (i.e. matching current epoch) do:

i. Set $\mathsf{sk}'_i = \mathsf{sk}_i + \delta_i$

ii. Append $(R, \mathsf{sk}'_i, \underline{\mathsf{seed}}, \mathsf{epoch})$ to rpool

iii. Remove this entry from fpool

6.7 Performance and Implementation

We discuss here the concrete overhead our refresh protocol adds to existing state of the art threshold ECDSA schemes, as most cryptocurrencies today (Bitcoin, Ethereum, etc.) use ECDSA as their canonical signature scheme. As at this point we are discussing specific protocols, we make the following observation: In the protocols of Lindell et al. [LNR18], Doerner et al. [DKLs19], and Gennaro and Goldfeder [GG18] the extra messages added by $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$ can be sent in parallel with the main ECDSA protocols. In particular, each $\pi^{\mathsf{R}}_{\mathrm{ECDSA}}$ has at least two rounds which can be used to generate K and δ in parallel, and each $\pi^{\sigma}_{\mathrm{ECDSA}}$ has at least one round before σ is released during which z can be constructed and verified.

6.7.1 Cost Analysis

In Table 6.7.1 we recall the costs of the $(\pi^{\mathsf{R}}_{\mathrm{ECDSA}}, \pi^{\sigma}_{\mathrm{ECDSA}})$ combined protocols of Doerner et al. [DKLs19] and Lindell et al. [LNR18] (OT-based) for perspective, and then give the overhead induced by $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$.

Protocol	Rounds	EC Mult.s	Comm.
Lindell et al. [LNR18]	8	239	195 KiB
Doerner et al. [DKLs19]	7	6	118 KiB
$\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$ **overhead**	0	6	192 Bytes

Table 6.7.1: Overhead of applying $\pi^{(2,n)}_{\rho\text{-}\mathsf{sign}}$ to proactivize $(2, n)$ ECDSA protocols instantiated with 256-bit curves. Figures are per-party and do not include cost of implementing proactive channels to communicate 160 bytes to each offline party every refresh.

Finally the update procedure $\pi^{(2,n)}_{\rho\text{-}\mathsf{update}}$ first requires reading the blockchain and scanning for signatures under the common public key since last awake– essentially the same operation as required to update balance of funds available in a wallet. Additionally one has to read messages received when offline and perform two curve multiplications for each refresh missed.

6.7.2 Implementation

In order to demonstrate the compatibility and efficiency of our refresh procedure, we implemented it to augment two different recent threshold ECDSA protocols; specifically those of Doerner et al. [DKLs19] and Gennaro and Goldfeder [GG18]. We present the results in this section.

We ran both sets of experiments on Amazon's AWS EC2 using a pair of t3.small machines located in the same datacenter for uniformity. However as the implementations of the base threshold ECDSA protocols came from different codebases, we stress that the important metric is the overhead added by our protocol in each case, and that comparison of the concrete times across the ECDSA protocols is not necessarily meaningful.

Proactivizing Doerner et al. [DKLs19]

As Doerner et al. natively utilize OT based multipliers, augmenting their threshold ECDSA signing with our refresh procedure yields a *fully* proactivized ECDSA wallet. We ran three experiments, during which we measured wall-clock time, including latency costs, collecting 100,000 samples and averaging them. We first ran their signing protocol unmodified, which took an average of 5.303ms to produce a signature. We then ran the same protocol augmented with our refresh generation procedure (i.e. $\pi^{(2,n)}_{\rho\text{-sign}}$) and found it to take an average of 6.587ms, i.e. a 24.2% increase. Finally we measured the cost of applying an update upon waking up (i.e. $\pi^{(2,n)}_{\rho\text{-update}}$) to be 0.381ms. Note that this figure does not account for the costs of the proactive channels or $\mathcal{G}_{\mathsf{Ledger}}$ (which is done anyway to update one's balance); the point of this benchmark is to demonstrate the efficiency of applying updates in isolation.

Gennaro and Goldfeder [GG18]

In order to understand the overhead added by the refresh procedure to the communication pattern of a different $(2, n)$ ECDSA based wallet, we implemented the protocol of Gennaro and Goldfeder [GG18] and augmented it with our refresh procedure during signing. Note their protocol makes use of a Paillier-based multiplier which we do not proactivize (see Canetti et al. [CGG+20] for how this can be done), and the cost of proactivizing an OT-based multiplier is negligible (0.381ms as shown previously). This is representative of the $(2, 3)$ cold storage application where the multipliers need not be offline-refreshed. We refer to the original $(\pi^{\mathsf{R}}_{\text{ECDSA}}, \pi^{\sigma}_{\text{ECDSA}})$ as GG and the augmented $\pi^{(2,n)}_{\rho\text{-sign}}$ as GG'.

We did not implement forward secure channels, we instead simulated it with reads from disk. We collected twenty samples for each configuration and found the average execution time of GG to be 1.433s and that of GG' to be 1.635s. In particular, $\pi^{(2,n)}_{\rho\text{-sign}}$ incurs a 14.09%

overhead in computation. Note that this figure does not include network latency, but in the LAN setting the measurements were within margin of error.

The code can be found in https://gitlab.com/neucrypt/mpecdsa/ (full proactivization of [DKLs19] by Jack Doerner) and https://github.com/KZen-networks/multi-party-ecdsa/tree/gg_pss (proactivization in KZen library).

6.8 General (t, n) Impossibility

We showed in Section 6.2.1 that an honest majority protocol is easy to construct, and so we assume for the rest of the discussion that we are in a setting where there is no online honest majority.

Many proactive secret sharing protocols in the literature have fundamentally followed the same approach: the refresh protocol runs roughly the same protocol that was used to share the secret, with new randomness incorporated to create an independent sharing of the same value. Therefore the ability to run verifiable secret sharing (VSS) in a given setting has always translated well to construct a refresh protocol for the same setting. Non-interactive VSS where only t online parties speak, with resiliency to $t-1$ corruptions are known in the literature [GMW91, Sta96] suggesting that their translation to our setting would yield an offline refresh protocol.

Unfortunately this intuition turns out to be false. Recall that a central principle in offline refresh is that all (honest) parties must be in agreement about whether or not to progress to the next epoch, i.e. 'unanimous erasure'. We discussed in Section 6.1 why anything less than this is undesirable, as even a simple network failure could induce permanent loss of the shared secret. However even this notion turns out to require the power of an honest majority to realize (barring the $(2, n)$ case) and we give intuition as to why below.

Recall that the refresh protocol π_ρ is run by t_ρ online parties, of whom $t-1$ may be corrupt, and we define $h = t_\rho - t + 1$ to denote how many are honest. Assume the weakest form of dishonest majority, i.e. one more corrupt party than honest, so $h = t-2$. The communication pattern of a single refresh phase is as follows: the online parties run π_ρ, following which each online party sends a message to each of the offline parties, who upon waking up will be able to catch up to the same epoch. The unanimous erasure property requires that all honest parties stay in agreement about the epoch; i.e. no one party is falsely convinced to prematurely erase their old state. Informally, we call a message or set of messages 'convincing' if they induce an offline party to progress to the next epoch and erase their old state.

Relating Unanimous Erasure to π_ρ It is instructive to view π_ρ as an MPC protocol to produce a convincing message for offline parties to progress. As we mandate unanimous erasure, it must never be the case that π_ρ permits an adversary to produce a convincing message while depriving online honest parties of it. In particular if π_ρ produces a convincing message then it must be visible and verifiable within the online honest parties' joint view (i.e. any subset of size h). Otherwise an adversary could at its discretion choose to induce an offline party to prematurely erase its state, and honest parties would not be able to tell either way. This property strongly suggests that π_ρ must achieve a form of *fairness* which does not bode well given that it must tolerate a dishonest majority.

A General Attack Now we hone in on exactly how an adversary can exploit the above facts. Assume that P_{off} is an offline party. Observe that the adversary is allowed to corrupt $h+1$ parties given the dishonest majority setting, and so it has the budget to keep h online parties corrupt as well as corrupt P_{off} initially, say in epoch 0. The adversary un-corrupts P_{off} and π_ρ is run successfully to move the system to epoch 1, keeping h parties corrupt (but behaving honestly) through the process. Now recall that the convincing message to P_{off} will be visible to any h online parties. Since the adversary has both: the state of P_{off} from epoch 0, as well as a 'convincing message' addressed to P_{off} by virtue of corrupting h parties during π_ρ, it is able to derive P_{off}'s refreshed state for epoch 1 despite not corrupting P_{off} in that epoch. Now simply corrupting one additional party in epoch 1 completely reveals the secret, as $h+2=t$ parties' private states are available to the adversary for that epoch.

Translating this intuition to a formal proof, or even a well-formed theorem, sees a number of subtle issues arise. For instance, we can not unconditionally prove that it is impossible to realize $\mathcal{F}^{n,t}_{\mathsf{ECDSA}}$ with offline refresh for $t > 2$; doing so would require proving that ECDSA itself is a signature scheme.[1]. To see why, consider a 'signature scheme' where the verification algorithm Vrfy outputs 1 on *all* inputs. Clearly, realizing a threshold version of this 'signature scheme', even with proactive security, is trivial; all parties simply output "0" when instructed to sign a message, then there is no private state to refresh. Therefore we formulate our theorem more carefully: we prove that if it possible to offline-refresh a given threshold signature scheme ($t > 2$) with a dishonest online majority, then the given signature scheme itself is succeptible to forgery.

We state our theorem in the $(\mathcal{G}_{\mathsf{Ledger}}, \mathcal{F}_{\mathsf{RO}})$ model, for the following reasons:

- $\mathcal{G}_{\mathsf{Ledger}}$ represents that this barrier can not be circumvented even with a consensus primitive as strong as an ideal ledger.

[1] At the moment ECDSA is known to be a signature only in the generic group model [Bro05], and not even in the random oracle model.

- $\mathcal{F}_{\mathsf{RO}}$ gives the power to compute any efficiently computable function [CLOS02] and so represents the ability to produce arbitrary correlated randomness during the preprocessing phase (i.e. during key generation) and also compute any function securely (albeit without robustness [Cle86]) during the refresh protocol itself.

Additionally both ideal oracles are trivial to implement when running the environment in a reduction.

Theorem 6.8.1. *Let* $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *be a triple of algorithms that satisfies the completeness definition of signature schemes. If there exists a protocol* $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$ *in the* $(\mathcal{G}_{\mathsf{Ledger}}, \mathcal{F}_{\mathsf{RO}})$*-hybrid model that UC-realizes* $\mathcal{F}^{n,t}_{\mathsf{Sign}}$ *with* $n > t_\rho \geq t > 2$ *in the presence of a mobile adversary actively corrupting* $t-1$ *parties where* $t_\rho < 2(t-1)$*, then there exists a forger for* Sig *that succeeds with overwhelming probability.*

Proof. We prove this theorem by first constructing an attack on the 'real' protocol $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$, and then using the simulator $\mathcal{S}_{\mathsf{Sign}}$ to translate this attack to the ideal protocol in order to construct a forger for Sig.

Consider an instantiation with parameters $n > t_\rho \geq t \geq 2$ such that $t_\rho < 2(t-1)$, i.e. less than half the parties in the refresh protocol are guaranteed to be honest. Define an experiment $\mathsf{EXEC}_{\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}, \mathcal{Z}}(1^\kappa)$ with environment $\mathcal{Z}_{n,t_\rho}$ as follows:

1. Send `init` to all parties.
2. Send $(\texttt{refresh}, [1, t_\rho])$ to each party P_i where $i \in [1, t_\rho]$.
3. Send $(\texttt{wake})$ to all parties.

All instructions are implemented with the protocol $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$. Let $\tau_{i,j}$ denote the transcript of the private channel from party P_i to P_j. Let state_i denote the private state of party P_i after the `init` command, and state'_i denote the private state of P_i after the $(\texttt{wake})$ command (note that state'_i is essentially the 'refreshed' state for the next epoch). Let 'off' index a canonical offline party, say $\mathsf{off} = t_\rho + 1$. Finally, let pk denote the public key produced when the `init` command is run.

We now show how to construct two algorithms: Ext to extract the state of P_{off} in an epoch of the protocol where it is not corrupted, and Sign^* that uses this state in conjunction with $t-1$ corrupt parties' states to sign any given message.

Lemma 6.8.2. *Define* $\tau_{i,j}, \mathsf{state}_i, \mathsf{state}'_i$ *for* $i, j \in [n]$*, and* off *as above, and let and* $h = t_\rho - t + 1$*. There is a pair of PPT algorithms* Ext *and* Sign^* *defined as follows:*

- Ext *:* $(\tau_{i,\mathsf{off}})_{i\in[h]}, \mathsf{state}_{\mathsf{off}} \mapsto \mathsf{state}'_{\mathsf{off}}$

- Sign^* *:* $m, \mathsf{state}'_{\mathsf{off}}, \{\mathsf{state}'_i\}_{i\in I} \mapsto \sigma$
  Where $I \subset [n] \setminus \{\mathsf{off}\}$ *and* $|I| = t-1$*, and* $m \in \{0,1\}^*$*.*

It holds that the following probability is overwhelming in κ*:*

$$\Pr\left[\mathsf{Vrfy}(\mathit{pk},\sigma,m)=1 : \begin{array}{rcl} \mathsf{state}'_{\mathsf{off}} & \leftarrow & \mathsf{Ext}((\tau_{i,\mathsf{off}})_{i\in[h]}, \mathsf{state}_{\mathsf{off}}) \\ \sigma & \leftarrow & \mathsf{Sign}^*(m, \mathsf{state}'_{\mathsf{off}}, \{\mathsf{state}'_i\}_{i\in I}) \end{array}\right]$$

Proof. In order to prove this lemma, we will show how to construct these algorithms.

First, some clarification on the parameters: Observe that since the maximum number of corruptions is $t-1$, the value $h = t_\rho - (t-1)$ represents the maximum guaranteed number of honest online parties in the refresh procedure. Additionally since $t > \lfloor t_\rho/2 \rfloor + 1$ it holds that the adversary may corrupt more than h parties. For ease of exposition, assume $2h+1 = t_\rho$ so that the adversary may corrupt up to $h+1$ parties and only h parties in the refresh protocol are honest in the worst case.

Consider the same experiment $\mathsf{EXEC}_{\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}, \mathcal{Z}^*}$ run with an alternative environment $\mathcal{Z}^*_{n,t_\rho}$ that corrupts each P_i for $i \in [h+1, 2h+1]$ and issues the same commands as $\mathcal{Z}_{n,t_\rho}$, with the caveat that corrupt parties do not transmit anything on their private channels to P_{off}, i.e. $(\tau_{i,\mathsf{off}} = \perp)_{i\in[h+1,2h+1]}$.

Observe that the view of the honest parties $P_1, \cdots, P_h$ is distributed identically in both executions. This is because the private channel between each corrupt P_i for $i \in [h+1, 2h+1]$ to P_{off} is hidden by definition, and P_{off} itself does not send any messages in this experiment. This fact has the following implications:

- The transcript of honest parties' private channels to P_{off}, i.e. $(\tau_{i,\mathsf{off}})_{i\in[h]}$ is distributed in both executions.

- The collection of private states of honest parties at the end of the experiment, i.e. $(\mathsf{state}'_i)_{i\in[h]}$, is distributed the same in both experiments. In particular, at the end of both experiments, parties $P_1, \cdots, P_h$ successfully advance to the next epoch. As all honest parties must agree on the epoch when activated, it holds that P_{off} advances to the next epoch in both experiments. In particular, for any $I \subset [n] \setminus \mathsf{off}$ such that $|I| = t-1$, it must hold that implementing the instruction $(\mathtt{sign}, m, I \cup \{\mathsf{off}\})$ via $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$ produces a valid signature σ of m under pk.

Note that the view of P_{off} is characterized entirely by the private channel communication from $P_1, \cdots, P_h$, i.e. $(\tau_{i,\mathsf{off}})_{i\in[h]}$ which is the same in both experiments, and $\mathsf{state}_{\mathsf{off}}$ its own private state from the start of the experiment (also the same in both experiments).

As we have argued that P_{off} must successfully advance to the next epoch in both experiments, we are ready to define Ext and Sign^* as follows:

- Ext implements the `wake` instruction for P_{off} via $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$, using as input the entire view of P_{off}, characterized by $(\tau_{i,\mathsf{off}})_{i\in[h]}, \mathsf{state}_{\mathsf{off}}$, and outputs the private state of P_{off} for the next epoch, $\mathsf{state}'_{\mathsf{off}}$.
- Sign^* implements the $(\texttt{sign}, m, I \cup \{\mathsf{off}\})$ instruction for $(P_i)_{i\in I}$ and P_{off} via $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$, using as input the private states of all of these parties $(\mathsf{state}'_i)_{i\in I\cup\{\mathsf{off}\}}$

By completeness and unanimous erasure of the protocol $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$, both the above algorithms succeed with overwhelming probability. This completes the proof of this lemma. □

We now construct the environment that will actually be used by the forger. Consider an instantiation with the same parameters as earlier, $n > t_\rho \geq t \geq 2$ such that $t > \lfloor t_\rho/2 \rfloor + 1$, i.e. less than half the parties in the refresh protocol are guaranteed to be honest, and define $\mathsf{off} = t_\rho + 1$ and $h = t_\rho - t + 1$ as earlier. Define the environment $\mathcal{Z}^*$ controlling adversary $\mathcal{A}$ as follows:

1. Instruct $\mathcal{A}$ to corrupt $P_1, P_2, \cdots, P_h$ and P_{off}.
2. Send `init` to all parties.
3. Instruct $\mathcal{A}$ to uncorrupt P_{off}.
4. Send $(\texttt{refresh}, [1, t_\rho])$ to each party P_i where $i \in [1, t_\rho]$.
5. Send $(\texttt{wake})$ to all parties.
6. Instruct $\mathcal{A}$ to corrupt P_{h+1}.
7. The adversary $\mathcal{A}$ outputs its entire view.
8. $\mathcal{Z}^*$ outputs whatever $\mathcal{A}$ outputs.

Note that unlike the usual specification for the real/ideal process in UC [Can01] in which the environment only outputs a bit, the output of $\mathcal{Z}^*$ here is a more complex string. This is done for ease of exposition as the output of $\mathcal{Z}^*$ will be used by the forger ($\mathcal{Z}^*$ acts as a passthrough for the output of $\mathcal{A}$), there is no meaningful advantage in the real/ideal distinguishing game.

Define $\tau_{i,j}, \mathsf{state}_i, \mathsf{state}'_i$ for $i, j \in [n]$ as earlier. The output of $\mathcal{A}$ at the end of this experiment is the complete views of parties $P_1, P_2, \cdots, P_h$, the view of P_{off} prior to the `refresh` instruction, and the view of P_{h+1} after the refresh instruction. These values are sufficiently characterized by $(\tau_{i,\mathsf{off}}, \mathsf{state}_i, \mathsf{state}'_i)_{i\in[h]}$, $\mathsf{state}_{\mathsf{off}}$, and state'_{h+1} respectively.

When the instructions of $\mathcal{Z}^*$ are implemented with the protocol $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$, we denote the output of the resulting experiment as $\mathsf{REAL}_{\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}},\mathcal{A},\mathcal{Z}^*}$. As $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$ UC-realizes $\mathcal{F}^{n,t}_{\mathsf{Sign}}$, there must exist a simulator $\mathcal{S}_{\mathsf{Sign}}$ which interacts with $\mathcal{Z}^*$ in place of $\mathcal{A}$, and queries $\mathcal{F}^{n,t}_{\mathsf{Sign}}$ instead of interacting with honest parties, with the output of the resulting experiment denoted $\mathsf{IDEAL}_{\mathcal{F}^{n,t}_{\mathsf{Sign}},\mathcal{S}_{\mathsf{Sign}},\mathcal{Z}^*}$. It must hold that $\mathsf{REAL}_{\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}},\mathcal{A},\mathcal{Z}^*} \approx \mathsf{IDEAL}_{\mathcal{F}^{n,t}_{\mathsf{Sign}},\mathcal{S}_{\mathsf{Sign}},\mathcal{Z}^*}$. We make use of this fact when constructing the forger, i.e. the forger will run the simulator $\mathcal{S}_{\mathsf{Sign}}$ with the adversary to sample from $\mathsf{IDEAL}_{\mathcal{F}^{n,t}_{\mathsf{Sign}},\mathcal{S}_{\mathsf{Sign}},\mathcal{Z}^*}$, as it can not sample from $\mathsf{REAL}_{\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}},\mathcal{A},\mathcal{Z}^*}$ without instantiating honest parties in $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$, for which their secret states (and hence the secret key) must be known. Additionally the challenger's public key pk can be embedded in the ideal computation using $\mathcal{F}^{n,t}_{\mathsf{Sign}}$.

We are finally ready to construct the forger for the signature scheme, which forges a signature on a given message m under a public key pk received from the challenger.

$\mathsf{Forge}(1^\kappa, \mathsf{pk}, m)$:

1. Sample $(\tau_{i,\mathsf{off}}, \mathsf{state}_i, \mathsf{state}'_i)_{i\in[h]}, \mathsf{state}_{\mathsf{off}}, \mathsf{state}'_{h+1} \leftarrow \mathsf{IDEAL}_{\mathcal{F}^{n,t}_{\mathsf{Sign}},\mathcal{S}_{\mathsf{Sign}},\mathcal{Z}^*}$ with the caveat that $\mathcal{F}^{n,t}_{\mathsf{Sign}}$ is programmed to output pk as the public key when `init` is queried by $\mathcal{S}_{\mathsf{Sign}}$. The ideal oracle $\mathcal{G}_{\mathsf{Ledger}}$ if used, is implemented as per its specification.
2. Compute $\mathsf{state}'_{\mathsf{off}} \leftarrow \mathsf{Ext}((\tau_{i,\mathsf{off}})_{i\in[h]}, \mathsf{state}_{\mathsf{off}})$
3. Compute $\sigma \leftarrow \mathsf{Sign}^*(m, \mathsf{state}'_{\mathsf{off}}, (\mathsf{state}'_i)_{i\in[h+1]})$
4. Output σ

Lemma 6.8.3. *For all $m \in \{0,1\}^*$, the following probability is overwhelming in κ:*

$$\Pr\left[\mathsf{Vrfy}(\mathit{pk},\sigma,m)=1 \;:\; \begin{array}{rl} (\mathit{sk},\mathit{pk}) \leftarrow & \mathsf{KeyGen}(1^\kappa) \\ \sigma \leftarrow & \mathsf{Forge}(\mathit{pk}, m)\end{array}\right]$$

Proof. We have previously shown in Lemma 6.8.2 that it is possible to forge a message under a public key pk' produced by running the real protocol $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$. We now show how to translate this ability in order to forge a message under a public key pk received from an external challenger (i.e. the signature experiment) using $\mathcal{S}_{\mathsf{Sign}}$ to replace honest parties from $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$ as well as program pk into the view of the adversary. We prove this lemma via a sequence of hybrid experiments.

Hybrid $\mathcal{H}_1$. In this hybrid experiment, Forge is run as specified, except that Step 1 is implemented using $\mathsf{REAL}_{\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}},\mathcal{A},\mathcal{Z}^*}$. Let the public key produced by running $\pi^{(t,n)}_{\rho\text{-}\mathsf{sign}}$ in REAL be pk'.

By Lemma 6.8.2, the output of Forge is a valid signature on m under pk' with overwhelming probability.

Hybrid $\mathcal{H}_2$. This hybrid experiment is the same as the last, except that Step 1 is implemented using $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{Sign}}^{n,t},\mathcal{S}_{\mathsf{Sign}},\mathcal{Z}^*}$ instead. As $\mathsf{REAL}_{\pi_{\rho\text{-}\mathsf{sign}}^{(t,n)},\mathcal{A},\mathcal{Z}^*} \approx \mathsf{IDEAL}_{\mathcal{F}_{\mathsf{Sign}}^{n,t},\mathcal{S}_{\mathsf{Sign}},\mathcal{Z}^*}$, the output of Forge is distributed indistinguishably to the last experiment (i.e. a valid signature under pk' chosen by $\mathcal{F}_{\mathsf{Sign}}^{n,t}$).

Hybrid $\mathcal{H}_3$. This hybrid experiment is the same as the last, with the caveat that $\mathcal{F}_{\mathsf{Sign}}^{n,t}$ is programmed to output pk as the public key when `init` is queried by $\mathcal{S}_{\mathsf{Sign}}$, instead of pk' that $\mathcal{F}_{\mathsf{Sign}}^{n,t}$ sampled internally. As pk and pk' are both sampled by running KeyGen with uniform randomness (by the challenger and $\mathcal{F}_{\mathsf{Sign}}^{n,t}$ respectively) it holds that $\{\mathsf{pk}\} \equiv \{\mathsf{pk}'\}$ which has the following implication:

$$
\begin{aligned}
&\Pr\left[\mathsf{Vrfy}(\mathsf{pk},\sigma,m)=1 : \begin{array}{rcl}(\mathsf{sk},\mathsf{pk}) &\leftarrow& \mathsf{KeyGen}(1^\kappa)\\ \sigma &\leftarrow& \mathcal{H}_3(m,\mathsf{pk})\end{array}\right]\\
&= \Pr\left[\mathsf{Vrfy}(\mathsf{pk}',\sigma',m)=1 : \begin{array}{rcl}(\mathsf{sk},\mathsf{pk}) &\leftarrow& \mathsf{KeyGen}(1^\kappa)\\ \sigma' &\leftarrow& \mathcal{H}_2(m,\mathsf{pk})\end{array}\right]\\
&= 1-\mathsf{negl}(\kappa)
\end{aligned}
$$

The final hybrid $\mathcal{H}_3$ is exactly the code of Forge, and outputs a valid signature on m under pk supplied by the challenger, which proves the lemma. □

The existence of an overwhelmingly successful forger for Sig given the existence of a protocol realizing $\mathcal{F}_{\mathsf{Sign}}^{n,t}$ with offline refresh, where $n > t_\rho \geq t > 2$, in the presence of a mobile adversary where $t > \lfloor t_\rho/2 \rfloor + 1$, is guaranteed by Lemma 6.8.3. The theorem is hence proven. □

6.9 Summary

With the increasing adoption of threshold wallets comes the need to defend them against mobile attackers. In this chapter we defined an "offline refresh" model for proactivizing threshold wallets with an optimal communication pattern, and studied0 this fine-grained notion of message complexity in the proactive setting.

We showed feasibility of honest majority offline refresh, and gave a comprehensive treatment of the dishonest majority setting: for the $(2, n)$ setting we devised a novel efficient protocol to proactivize many standard signature schemes with offline refresh, and implemented

it to show that it adds little overhead in practice. Finally we showed that it is impossible to have the refresh protocol tolerate a dishonest majority of participants, without having all parties come online at least at some point in each epoch. We developed new techniques to prove this theorem, and believe that they will find application in reasoning about proactive security in other contexts. However there may be relaxations of the model, physical hardware assumptions, or nonstandard trust models that are still reasonable in practice; we leave open the problem of identifying such models and tailoring constructions for them.

Chapter 7

Conclusion and Future Work

Managing cryptographic keys is a notoriously difficult problem in practice, as they frequently induce single-point-of-failure vulnerabilities in deployed cryptosystems. Threshold cryptography, which draws from the rich theory of Secure Multiparty Computation, presents a promising method by which to mitigate the single point of failure problem. However, when designing systems and protocols for real-world use, issues that have not yet been adequately accounted for in the literature are bound to arise. In this thesis we explored a few such issues, in particular those relating to state continuity, availability of reliable entropy, interaction patterns, and long-term security.

Our results in this thesis present new conceptual methods to address some of these issues for the two most commonly deployed elliptic curve signature schemes—ECDSA and EdDSA–with empirical or concretely estimated justification of their practicality.

In Chapter 4 we explored how to non-interactively aggregate EdDSA signatures with computational efficiency that induces tolerable latency for most relevant applications. There is ample room to improve the efficiency of our tightly secure construction, and we leave as an interesting open problem how our techniques can be extended to aggregate ECDSA signatures. We additionally showed Fischlin's transformation to be insecure when applied to certain common Sigma protocols, and traced the issue to its deterministic nature. We showed how to patch this insecurity by means of careful randomization, and leave for future work to investigate the necessity of randomization when compiling Sigma protocols to NIZKPoKs with straight-line extraction in the random oracle model.

It is well known that reliable entropy is scarce in practice, which in combination of a high risk of state reuse in many contexts leads to unique vulnerabilities for distributed signing with ECDSA/EdDSA. We studied this issue in Chapter 5 and provided a solution for EdDSA based on tools native to EdDSA itself, that we estimate will in many settings induce a lower latency than heuristic solutions with trusted hardware. Most practically efficient distributed

ECDSA/EdDSA protocols today do not account for state incontinuity in conjunction with poor online entropy, however the tools to solve this problem do exist in the literature—albeit not necessarily efficient enough for many applications. The design of efficient derandomized threshold signing protocols that are naturally resilient to state reuse therefore represents a fruitful direction for future research, perhaps by optimizing MPC and zero-knowledge techniques for the relevant statements.

Finally in Section 3.4, we argued that the interaction patterns induced by most protocols to refresh the state of threshold cryptosystems today (as a long-term defense mechanism), are not ideal for many threshold cryptosystems. We gave an empirically justified solution in Chapter 6 for the $(2, n)$ ECDSA/EdDSA setting in the context of cryptocurrency wallets, along with some barriers to extending the techniques. We leave as future research how to circumvent our bounds while still achieving meaningful notions of security.

Overall, we presented several concrete directions for future work in each chapter, with many open questions around the theory and practical efficiency of distributed signing (and related tools such as zero-knowledge proofs). Designing distributed signing protocols for ECDSA and EdDSA that are sensitive to the practical constraints of a given context, can serve to significantly enhance the security of many deployed cryptosystems that rely upon them.

Bibliography

[ABGR13] Prabhanjan Ananth, Raghav Bhaskar, Vipul Goyal, and Vanishree Rao. On the (in)security of Fischlin's paradigm. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 202–221. Springer, Heidelberg, March 2013.

[ACD19] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. In *EUROCRYPT 2019*, pages 129–158, 2019.

[ADN06] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold rsa with adaptive and proactive security. In *EUROCRYPT '06*, pages 593–611, 2006.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

[AMM+] David Archer, Victor Arribas Abril Pieter Maene, Nele Mertens, Danilo Sijacic, and Nigel Smart. Bristol fashion mpc circuits. https://homes.esat.kuleuven.be/~nsmart/MPC/. Accessed: Aug 14, 2021.

[AMMR18] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1993–2010. ACM Press, October 2018.

[ANT+20] Diego F. Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. Ladderleak: Breaking ecdsa with less than one bit of nonce leakage. Cryptology ePrint Archive, Report 2020/615, 2020. https://eprint.iacr.org/2020/615.

[Bay14] James Bayer. Challenges With Randomness In Multi-tenant Linux Container Platforms, 2014.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE S&P*, pages 315–334. IEEE Computer Society Press, May 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[BCJZ21] Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The provable security of ed25519: Theory and practice. In *IEEE S&P 2021*, 2021.

[BCKL21] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve fast fourier transform (ECFFT) part I: fast polynomial algorithms over all finite fields. *ECCC*, page 103, 2021.

[BCLK17] Marcus Brandenburger, Christian Cachin, Matthias Lorenz, and Rüdiger Kapitza. Rollback and forking detection for trusted execution environments using lightweight collective memory. In *DSN 2017*, 2017.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, September 2012.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464. Springer, Heidelberg, December 2018.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In *CRYPTO '95*, pages 97–109, 1995.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 2025–2038. ACM, 2020.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020*, 2020.

[BGG+20] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 260–290. Springer, 2020.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BHH+19] Michael Backes, Lucjan Hanzlik, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 286–313. Springer, Heidelberg, April 2019.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

[Blo] Average transactions per block — blockchain.com. https://www.blockchain.com/charts/n-transactions-per-block. Accessed: 2022-Feb-11.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

[Blu86] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, volume 1, page 2. Citeseer, 1986.

[BM90] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 547–557. Springer, Heidelberg, August 1990.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO 2017*, pages 324–356, 2017.

[Bro05] Daniel R. L. Brown. Generic groups, collision resistance, and ECDSA. *Des. Codes Cryptography*, 2005.

[BSGL20] Eli Ben-Sasson, Lior Goldberg, and David Levit. Stark friendly hash – survey and recommendation. *IACR Cryptol. ePrint Arch.*, 2020:948, 2020.

[BST21] Charlotte Bonte, Nigel P. Smart, and Titouan Tanguy. Thresholdizing hashed-dsa: MPC to the rescue. *International Journal of Information Security*, 2021.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCD+20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and Abhi Shelat. Multiparty generation of an RSA modulus. In *CRYPTO 2020*, 2020.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019.

[CCL+19] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, pages 191–221, 2019.

[CD00] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 331–345. Springer, Heidelberg, December 2000.

[CDD+15] Ignacio Cascudo, Ivan Damgård, Bernardo Machado David, Irene Giacomelli, Jesper Buus Nielsen, and Roberto Trifiletti. Additively homomorphic UC commitments with optimal amortized overhead. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 495–515. Springer, Heidelberg, March / April 2015.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

[CGG+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1769–1787. ACM, 2020.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *32nd ACM STOC*, pages 235–244. ACM Press, May 2000.

[CGGN17] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 229–243. ACM Press, October / November 2017.

[CGKN21] Konstantinos Chalkias, François Garillot, Yashvanth Kondi, and Valeria Nikolaenko. Non-interactive half-aggregation of eddsa and variants of schnorr signatures. In Kenneth G. Paterson, editor, *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, pages 577–608. Springer, 2021.

[CGM16] Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 499–530. Springer, Heidelberg, August 2016.

[CGN20] Konstantinos Chalkias, François Garillot, and Valeria Nikolaenko. Taming the many eddsas. In Thyla van der Merwe, Chris J. Mitchell, and Maryam Mehrnezhad, editors, *Security Standardisation Research - 6th International Conference, SSR 2020, London, UK, November 30 - December 1, 2020, Proceedings*, volume 12529 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 2020.

[CH94] Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 425–438. Springer, Heidelberg, August 1994.

[CHH00] Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. *J. Cryptology*, 13(1):61–105, 2000.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, November 2014.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, March 2012.

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 88–97, 2002.

[Cle86] R Cleve. Limits on the security of coin flips when half the processors are faulty. In *STOC '86*, 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 494–503. ACM, 2002.

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 40–58. Springer, Heidelberg, August 2015.

[CZ22] Yanbo Chen and Yunlei Zhao. Half-aggregation of schnorr signatures with tight reductions. Cryptology ePrint Archive, Paper 2022/222, 2022. https://eprint.iacr.org/2022/222.

[Dam02] Ivan Damgård. On Σ-protocols. In *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[DDGN14] Ivan Damgård, Bernardo Machado David, Irene Giacomelli, and Jesper Buus Nielsen. Compact VSS and efficient homomorphic UC commitments. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 213–232. Springer, Heidelberg, December 2014.

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Heidelberg, August 1988.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE S&P*, pages 980–997. IEEE Computer Society Press, May 2018.

[DKLs19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE S&P*, pages 1051–1066. IEEE Computer Society Press, May 2019.

[DLS88] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 581–596. Springer, Heidelberg, August 2002.

[DOK+20] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. Securing DNSSEC keys via threshold ECDSA from generic MPC. In *ESORICS 2020*, 2020.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[EOPY18] Karim Eldefrawy, Rafail Ostrovsky, Sunoo Park, and Moti Yung. Proactive secure multiparty computation with a dishonest majority. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2018.

[Eya21] Ittay Eyal. On cryptocurrency wallet design. In *3rd International Conference on Blockchain Economics, Security and Protocols*. 2021.

[EZJ+14] Adam Everspaugh, Yan Zhai, Robert Jellinek, Thomas Ristenpart, and Michael Swift. Not-So-Random Numbers in Virtualized Linux and the Whirlwind RNG. In *2014 IEEE Symposium on Security and Privacy*, pages 559–574. IEEE, may 2014.

[Fel87a] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, pages 427–437. IEEE Computer Society Press, October 1987.

[Fel87b] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles,*

California, USA, 27-29 October 1987, pages 427–437. IEEE Computer Society, 1987.

[FGH+02] Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein, and Adam D. Smith. Detectable byzantine agreement secure against faulty majorities. In Aleta Ricciardi, editor, *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing, PODC 2002, Monterey, California, USA, July 21-24, 2002*, pages 118–126. ACM, 2002.

[FGMY97] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Proactive rsa. In Burton S. Kaliski, editor, *CRYPTO '97*, 1997.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, August 2005.

[FJNT16] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. On the complexity of additively homomorphic UC commitments. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 542–565. Springer, Heidelberg, January 2016.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 191–219. Springer, Heidelberg, April 2015.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990.

[Gav21] Jiří Gavenda. *Threshold ECDSA Performance Analysis*. Bachelor's thesis. Masaryk University, Faculty of Informatics, 2021.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1179–1194, 2018.

[GIKW14] Juan A. Garay, Yuval Ishai, Ranjit Kumaresan, and Hoeteck Wee. On the complexity of UC commitments. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 677–694. Springer, Heidelberg, May 2014.

[Gil99] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129. Springer, Heidelberg, August 1999.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT 2015*, pages 281–310, 2015.

[GKM+20] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. Cryptology ePrint Archive, Report 2020/504, 2020. https://eprint.iacr.org/2020/504.

[GKMN21] François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. Threshold schnorr with stateless deterministic signing from standard assumptions. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 127–156. Springer, 2021.

[GKPS18] Chaya Ganesh, Yashvanth Kondi, Arpita Patra, and Pratik Sarkar. Efficient adaptively secure zero-knowledge from garbled circuits. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 499–529. Springer, Heidelberg, March 2018.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, July 2005.

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 567–578. ACM Press, October 2015.

[GLSY04] Rosario Gennaro, Darren Leigh, R. Sundaram, and William S. Yerazunis. Batching Schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 276–292. Springer, Heidelberg, December 2004.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, July 1991.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 101–111. ACM, 1998.

[Hen22] Nadia Heninger. RSA, DH and DSA in the Wild. In Joppe Bos and Martijn Stam, editors, *Computational Cryptography*, chapter 6, pages 140–181. Cambridge University Press, 2022.

[HJJ+97] Amir Herzberg, Markus Jakobsson, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In *CCS '97, Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, April 1-4, 1997.*, pages 100–110, 1997.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith,

editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 339–352. Springer, Heidelberg, August 1995.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In *EUROCRYPT*, 2020.

[HMPs14] Susan Hohenberger, Steven Myers, Rafael Pass, and abhi shelat. ANONIZE: A large-scale anonymous survey system. In *2014 IEEE S&P*, pages 375–389. IEEE Computer Society Press, May 2014.

[HS01] Nick Howgrave-Graham and Nigel P. Smart. Lattice attacks on digital signature schemes. *Des. Codes Cryptogr.*, 23(3):283–290, 2001.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 598–628. Springer, Heidelberg, December 2017.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO '03*, pages 145–161, 2003.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966. ACM Press, November 2013.

[KASN15] Rashmi Kumari, Mohsen Alimomeni, and Reihaneh Safavi-Naini. Performance Analysis of Linux RNG in Virtualized Environments. In *ACM Workshop on Cloud Computing Security Workshop - CCSW '15*, New York, New York, USA, 2015.

[KC12] Brendan Kerrigan and Yu Chen. A Study of Entropy Sources in Cloud Computers: Random Number Generation on Cloud Hosts. pages 286–298. 2012.

[KL15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*, chapter Digital Signature Schemes, pages 443–486. Chapman & Hall/CRC, 2015.

[KL17] Dmitry Khovratovich and Jason Law. BIP32-Ed25519: Hierarchical Deterministic Keys over a Non-linear Keyspace. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 27–31. IEEE, IEEE, apr 2017.

[KMOS21] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 608–625. IEEE, 2021.

[Kos] Ahmed Kosba. xJsnark.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *CRYPTO*, 2015.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 830–842. ACM, 2016.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.

[Kra93] D.W. Kravitz. Digital signature algorithm, jul 1993. US Patent 5,231,668.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 365–391. Springer, Heidelberg, August 2018.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

[Ks22] Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. *IACR Cryptol. ePrint Arch.*, page 393, 2022.

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 2003*, pages 155–164. ACM Press, October 2003.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multiparty computation using a global transaction ledger. In *EUROCRYPT 2016*, pages 705–734, 2016.

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 446–466. Springer, Heidelberg, May 2011.

[Lin17] Yehuda Lindell. Fast secure two-party ecdsa signing. In *CRYPTO*, 2017.

[Lin22] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. *IACR Cryptol. ePrint Arch.*, page 374, 2022.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 74–90. Springer, Heidelberg, May 2004.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1837–1854. ACM Press, October 2018.

[LNR18] Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. *IACR Cryptology ePrint Archive*, 2018:987, 2018.

[LPR19] Yehuda Lindell, Guy Peer, and Samuel Ranellucci. Unbound blockchain-crypto-mpc library. *White Paper*, 2019.

[LSTW21] Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Linear-time zero-knowledge snarks for R1CS. *IACR Cryptol. ePrint Arch.*, page 30, 2021.

[MAK+17] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback protection for trusted execution. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 1289–1306. USENIX Association, August 2017.

[MH20] Gabrielle De Micheli and Nadia Heninger. Recovering cryptographic keys from partial information, by example. *IACR Cryptol. ePrint Arch.*, 2020:1506, 2020.

[MNPV99] David M'Raïhi, David Naccache, David Pointcheval, and Serge Vaudenay. Computational alternatives to random number generators. In Stafford E. Tavares and Henk Meijer, editors, *SAC 1998*, volume 1556 of *LNCS*, pages 72–80. Springer, Heidelberg, August 1999.

[MP] Moxie Marlinspike and Trevor Perrin. The double ratchet algorithm, 11 2016. In *https://signal.org/docs/specifications/x3dh/x3dh.pdf*.

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.

[MZW+19] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. CHURP: Dynamic-committee proactive secret sharing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2369–2386. ACM Press, November 2019.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[NIS] NIST Computer Security Division, Information Technology Laboratory. Multi-party threshold cryptography: Computer security resource center. https://csrc.nist.gov/Projects/threshold-cryptography. (Date accessed: 11-17-2021).

[NKDM03] Antonio Nicolosi, Maxwell N. Krohn, Yevgeniy Dodis, and David Mazières. Proactive two-party signatures for user authentication. In *NDSS 2003*. The Internet Society, February 2003.

[NN05] Ventzislav Nikov and Svetla Nikova. On proactive secret sharing schemes. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography*, pages 308–325, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[NRSW20] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. *ACM CCS 2020*, 2020.

[NSW09] Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for schnorr signatures. *J. Math. Cryptol.*, 3(1):69–87, 2009.

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Heidelberg, August 1993.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *10th ACM PODC*, pages 51–59. ACM, August 1991.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337. Springer, Heidelberg, August 2003.

[Ped91] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 522–526. Springer, Heidelberg, April 1991.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[PLD+11] Bryan Parno, Jacob R. Lorch, John R. Douceur, James W. Mickens, and Jonathan M. McCune. Memoir: Practical state continuity for protected modules. In *2011 IEEE S&P*, pages 379–394. IEEE Computer Society Press, May 2011.

[Pre93] Bart Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit te Leuven, 1993.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. *CRYPTO 2020*, 2020.

[SLdW07] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 1–22. Springer, Heidelberg, May 2007.

[SP16] Raoul Strackx and Frank Piessens. Ariadne: A minimal approach to state continuity. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 875–892. USENIX Association, August 2016.

[SS01] Douglas R. Stinson and Reto Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In Vijay Varadharajan and Yi Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 417–434. Springer, Heidelberg, July 2001.

[Sta96] Markus Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pages 190–199, 1996.

[TTA18] Akira Takahashi, Mehdi Tibouchi, and Masayuki Abe. New Bleichenbacher records: Fault attacks on qDSA signatures. *IACR TCHES*, 2018(3):331–371, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7278.

[TZ21] Akira Takahashi and Greg Zaverucha. Verifiable encryption from mpc-in-the-head. *IACR Cryptol. ePrint Arch.*, page 1704, 2021.

[Unr15] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 755–784. Springer, Heidelberg, April 2015.

[vDRSD07] Marten van Dijk, Jonathan Rhodes, Luis F. G. Sarmenta, and Srinivas Devadas. Offline untrusted storage with immediate detection of forking and replay attacks. In *ACM STC '07*, 2007.

[vM39] Richard von Mises. *Über Aufteilungs-und Besetzungswahrscheinlichkeiten*. na, 1939.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3 edition, 2013.

[wbo] Ecfft algorithms on the bn254 base field. https://github.com/wborgeaud/ecfft-bn254. Accessed: 2022-Feb-12.

[Wha21] Zander Wharton. Multi-factor authentication adoption: 75managers plan to increase mfa spending according to new study by yubico and 451 research. https://www.yubico.com/blog/75-of-enterprise-security-managers-plan-to-increase-mfa-spending-according-to-new-study-by-yubico-and-451-research/, 2021. Accessed: May 7, 2022.

[Woo] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

Appendix A

Postponed Proofs of Straight-Line Extraction Theorems

A.1 Full Proof of Theorem 4.4.5

We first define r-special sound sigma protocols.

Definition A.1.1. *Let κ be the security parameter, which is polynomially related to the instance size. A strongly r-special sound Sigma protocol for relation R is a three move public coin protocol between a prover P_Σ and verifier V_Σ that has the following properties:*

- ***Completeness**: If P_Σ (with private input w) and V_Σ with public input x such that $(x, w) \in R$ execute the protocol honestly, then the protocol always terminates in $\mathsf{poly}(\kappa)$ time with V accepting.*
- ***Strong r-special soundness**: There exists a PPT extractor Ext which given as input the accepting conversations $\{T_i = (a, e_i, z_i)\}_{i\in[r]}$ for statement x such that $T_i \neq T_j$ for every distinct pair $i, j \in [r]$, outputs w such that $(x, w) \in R$.*
- ***Honest verifier zero-knowledge/$r-1$ Simulatability**: There exists a PPT simulator $\mathcal{S}$ which upon input a statement x and challenges $\{e_i\}_{i\in[r-1]}$ outputs $a, \{z_i\}_{i\in[r-1]}$ such that each (a, e_i, z_i) is an accepting conversation.*

We restate the theorem below, and give the full proof.

Theorem A.1.2. *If Σ is a strongly $r+1$-special sound Sigma protocol and $\ell(r-1) = \kappa$, the protocol π_{NIZK} is a straight-line extractable NIZKPoK in the random oracle model, with an extractor that does not program the random oracle and achieves extraction error $Q/2^\kappa$ for an adversary making Q queries to the random oracle.*

Extractor $\mathsf{Ext}_{\mathsf{NIZK}}$

The extractor is given the statement x, a proof π, and the list of queries to the random oracle $\boldsymbol{Q}$ that were made by the adversary in the production of this proof. In addition to this, this extractor has access to the extractor Ext_{Σ} of the strongly $r+1$ special sound sigma protocol, which requires $r+1$ accepting transcripts (with the same a value) in order to produce a witness w for the statement.

$\mathsf{Ext}_{\mathsf{NIZK}}(x, \pi, \boldsymbol{Q})$:

1. Parse $(\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z}) = \pi$, and $(e_i)_{i\in[r]} = \boldsymbol{e}$, and $(z_i)_{i\in[r]} = \boldsymbol{z}$
2. Initialize $\tau = (e_i, z_i)_{i\in[r]}$
3. Search $\boldsymbol{Q}$ until a query of the form $(\boldsymbol{a}, e, z)$ is found such that $(e, z) \notin \tau$, and $\mathsf{V}_{\Sigma}(x, (\boldsymbol{a}, e, z)) = 1$
4. Output $\mathsf{Ext}_{\Sigma}(a_i, \tau)$

Figure A.1: Extracting a witness

Proof. We first argue completeness, then extraction and zero-knowledge.

Completeness: The prover P terminates successfully with a proof when it finds a multicollision of size r for a function that maps a domain of size $r \cdot 2^{\ell}$ to a range of size 2^{ℓ}. By the pigeonhole principle, there exists at least one such multicollision, and since the prover checks the domain exhaustively, such a multicollision is always found.

Extraction: We give the straight-line extractor $\mathsf{Ext}_{\mathsf{NIZK}}$ in Figure A.1 and then argue that it fails with probability exponentially small in κ.

The event in which this extractor fails is the event in which an adversarial prover P^* is able to produce a proof π by querying no more than r accepting Sigma protocol transcripts to the random oracle. We first ignore all queries made to H that are not accepting transcripts, and then separate queries prefixed by different $\boldsymbol{a}$ as they essentially instantiate independent random oracles (and are not compatible with one another). For a given $\boldsymbol{a}$, the event in which the adversary is able to output an accepting proof with fewer than $r+1$ accepting transcripts (prefixed by $\boldsymbol{a}$) queried to H is exactly the event that all of the first r such accepting transcripts queried to H evaluate to the same value. This is equivalent to r independent uniformly chosen ℓ-bit strings being equal, which happens with probability $(2^{-\ell})^{(r-1)} = 2^{-\kappa}$. For an adversary that makes Q queries to the random oracle, the extraction error is therefore bounded by $Q/2^{\kappa}$.

Zero-knowledge: We describe how to simulate a proof in Figure A.2, and then show its

Simulator $\mathcal{S}_{\mathsf{NIZK}}$

Simulator $\mathcal{S}_{\mathsf{NIZK}}$ is given the statement x, and has the ability to program the Random Oracle H. In addition to this $\mathcal{S}_{\mathsf{NIZK}}$ is given the simulator for the Sigma protoocol $\mathcal{S}_{\Sigma}$. Let $t = \ell + \log r$

$\mathcal{S}_{\mathsf{NIZK}}(x)$:

1. Uniformly sample $e_i \leftarrow \{0,1\}^t$ for each $i \in [r]$ and set $\boldsymbol{e} = (e_i)_{i\in[r]}$
2. Run the simulator for the sigma protocol to obtain $(\boldsymbol{a}, \boldsymbol{z}) \leftarrow \mathcal{S}_{\Sigma}(x, r, \boldsymbol{e})$
3. Sample $v \leftarrow \{0,1\}^{\ell}$
4. Program the random oracle H so that $H(\boldsymbol{a}, e_i, z_i) = v$ for each $i \in [r]$
5. Emulate H as a random oracle 'honestly' for every other query
6. Output $\pi = (\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z})$

Figure A.2: Simulator for Zero-Knowledge

indistinguishability from a real proof.

We argue that the simulation is indistinguishable from a real proof through a sequence of hybrid experiments, which are defined as follows.

Hybrid $\mathcal{H}_1$. The real prover's algorithm (P from π_{NIZK}) is used to find $(\mathbf{a}, \mathbf{e}, \mathbf{z})$ such that $H(\boldsymbol{a}, e_1, z_1) = \cdots = H(\boldsymbol{a}, e_r, z_r)$ where H is emulated as a random oracle by the standard technique of maintaining a (query, response) table. The difference from the real prover's algorithm is merely syntactic.

Hybrid $\mathcal{H}_2$. Implement Steps 3 and 4 of $\mathcal{S}_{\mathsf{NIZK}}$. In particular in this experiment, the random oracle H is implemented as follows:

1. Sample $v \leftarrow \{0,1\}^{\ell}$
2. The first r queries by the honest prover $Q_1, Q_2, \cdots Q_r$ (where each $Q_i = (\mathbf{a}, e_i, z_i)$ as generatred by P) will receive v as a response, i.e. $H(Q_1) = H(Q_2) = \cdots = H(Q_k) = v$
3. Emulate H as a random oracle 'honestly' for every other query

This hybrid differs from the last in that here the prover P will terminate after the first r queries it makes to H, whereas in $\mathcal{H}_1$ since H is not programmed to shortcut to a multicollision, P will have to 'work' to find a multicollision. Since the difference in running time of

$\mathcal{H}_2$ and $\mathcal{H}_1$ is invisible to a distinguisher and $\boldsymbol{a}$ are generated identically in both hybrids, the only component that remains to be analyzed is $\boldsymbol{e}$ (since $\boldsymbol{z}$ is implicitly fixed by $w, \boldsymbol{a}, \boldsymbol{e}$). In $\mathcal{H}_1$, $\boldsymbol{e}$ represents the indices of the first multicollision found by P relative to H. Since P steps through pre-images uniformly at random and H is a random oracle (i.e. H has independent uniformly random outputs for every pair of distinct inputs) $\boldsymbol{e}$ is distributed uniformly in $\{0,1\}^{t \times r}$ in $\mathcal{H}_1$. In $\mathcal{H}_2$, $\boldsymbol{e}$ is clearly uniformly distributed in $\{0,1\}^{t \times r}$ as it corresponds to the first r challenges tried by P, which are sampled uniformly and independently.

As $\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z}$ are distributed identically in $\mathcal{H}_2$ and $\mathcal{H}_1$, the only distinguishing event corresponds to the programming of H, i.e. if the adversary is able to query H on some index that $\mathcal{H}_2$ subsequently programs to a different value. Since $\boldsymbol{a}$ has at least κ bits of entropy and is a prefix for all queries programmed in $\mathcal{H}_2$, this distinguishing event happens with probability no greater than $Q/2^{\kappa}$, where Q is the number of queries made by the adversary to the random oracle.

Hybrid $\mathcal{H}_3$. Replace the role of P in generating $(\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z})$ by Steps 1 and 2 of $\mathcal{S}_{\mathsf{NIZK}}$. In particular while $\mathcal{H}_2$ computes $\boldsymbol{a}, \mathsf{state} \leftarrow P_{\Sigma,a}(w)$, samples each challenge $e_i \leftarrow \{0,1\}^{t \times r}$, and produces each $z_i \leftarrow P_{\Sigma,z}(\mathsf{state}, e_i)$, this hybrid simply computes $(\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z}) \leftarrow \mathcal{S}_{\mathsf{NIZK}}(x)$. This modification still retains perfect correctness, as $\mathcal{H}_2$ already programs H to 'shortcut' to a multicollision upon being queried on each $(\boldsymbol{a}, e_i, z_i)$ produced. Indistinguishability of $(\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{z})$ produced in $\mathcal{H}_3$ and $\mathcal{H}_2$ directly follows from r-simulatability of the Sigma protocol; there is a trivial lossless reduction to translate a distinguisher for $\mathcal{H}_3$ and $\mathcal{H}_2$ to a distinguisher for r-simulatibility of the Sigma protocol.

The final hybrid experiment $\mathcal{H}_2$ implements the simulator $\mathcal{S}_{\mathsf{NIZK}}$ in its entirety, and does not take the witness w as an input. As we show that for any $(x, w) \in R$, it holds that $\mathsf{P}^H(w, x) \equiv \mathcal{H}_0(w, x) \approx \mathcal{H}_3(x) \equiv \mathcal{S}_{\mathsf{NIZK}}(x)$, zero-knowledge of π_{NIZK} is hence proven.

□

A.2 Strongly r-special Sound Schnorr

It is easy to modify Schnorr's proof of knowledge of discrete logarithm protocol [Sch91] to an r-special sound Sigma protocol with $r-1$-simulatability. This is achieved (in spirit) by instantiating the batched Schnorr protocol of Gennaro et al. [GLSY04] where one 'batches' $r-1$ random instances with the given instance. Intuitively in order to prove knowledge of the discrete log $x \in \mathbb{Z}_q$ of a public $X \in \mathbb{G}$ (where $\mathbb{G}$ is say an elliptic curve group), the prover samples a random degree $r-1$ polynomial $f \in \mathbb{Z}_q[X]$ such that $f(0) = x$, and publishes

Protocol $\Sigma_{r,\mathsf{DL}}$

The prover $\mathsf{P} = (P_{\Sigma,a}, P_{\Sigma,z})$ and verifier V are both given public parameters $(\mathbb{G}, G, q)$, $r \in \mathbb{Z}$, and the statement $X = x \cdot G$. The prover additionally has witness x as private input.

$P_{\Sigma,a}(X, x)$:

1. Sample $r-1$-degree polynomial $f \in \mathbb{Z}_q[X]$ such that $f(0) = x$
2. Compute commitment $\boldsymbol{a} = (f(i) \cdot G)_{i \in [r-1]}$, and set $\mathsf{state} = f$
3. Output $(\mathsf{state}, \boldsymbol{a})$

$P_{\Sigma,z}(\mathsf{state}, e)$:

1. Parse $e \in \mathbb{Z}_q^*$ and output $f(e)$

$\mathsf{V}(X, \boldsymbol{a}, e, z)$:

1. Parse $a_1, a_2, \cdots, a_r = \boldsymbol{a}$
2. Define degree $r-1$ polynomial $F \in \mathbb{G}[X]$ such that $F(0) = X$ and $F(i) = a_i$
3. Output $F(e) \stackrel{?}{=} z \cdot G$

Figure A.3: r-special sound proof of Discrete Log

$\boldsymbol{a} = (f(i) \cdot G)_{i \in [r-1]}$. Given a challenge $e \in \mathbb{Z}_q^*$, the prover reveals $f(e)$, which the verifier can check is indeed the discrete logarithm of $F(e)$ by interpolation in the exponent, where $F \in \mathbb{G}[X]$ is the degree $r-1$ polynomial such that $F(0) = X$ and $\{F(i) = \boldsymbol{a}_i\}_{i \in [r-1]}$.

We give the protocol $\Sigma_{r,\mathsf{DL}}$ in Figure A.3.

Theorem A.2.1. *The protocol $\Sigma_{r,\mathsf{DL}}$ is a strongly r-special sound Sigma protocol for the language DLog.*

Proof. Completeness is easy to verify. $r-1$-simulatability and r-special soundness are discussed below.

$r-1$-simulatability. Transcript $\boldsymbol{a}, (z_i)_{i \in [r-1]}$ can be simulated given $X = g^x$ and $e_1, e_2, \cdots, e_r - 1$ as follows:

1. Sample $z_i \leftarrow \mathbb{Z}_q$ and compute $Z_i = z_i \cdot G$ for each $i \in [r-1]$
2. Define degree $r-1$ polynomial $F \in \mathbb{G}[X]$ such that $F(0) = X$ and $F(e_i) = Z_i$

3. Compute $\boldsymbol{a} = \{F(i)\}_{i\in[r-1]}$

4. Output $\boldsymbol{a}, (z_i)_{i\in[r-1]}$

The real prover samples f by choosing $\{f(i)\}_{i\in[r-1]}$ uniformly, and publishes $\boldsymbol{z} = \{f(e_i)\}_{i\in[r-1]}$ which is effectively uniform in $\mathbb{Z}_q^r$. The simulator chooses uniform $\boldsymbol{z} = \{f(e_i)\}_{i\in[r-1]}$ directly, and so $\boldsymbol{z}$ is distributed identically in both executions. As $\mathbb{Z}_q$ is isomorphic to $\mathbb{G}$, the $\boldsymbol{a}$ values are fixed given $X, \boldsymbol{z}$, which accounts for all components in the view and proves that the simulated and real values are identically distributed.

Strong r-special soundness. Given r accepting transcripts (ie. correct polynomial evaluations), by the facts that there can exist at most one $r-1$-degree polynomial passing through r points that $\mathbb{Z}_q$ and $\mathbb{G}$ are isomorphic, the points $(e_1, z_1), (e_2, z_2), \cdots, (e_r, z_r)$ fully specify $f \in \mathbb{Z}_q[X]$ such that $\{f(i) \cdot G = a_i\}_{i\in[r-1]}$ and $f(0) \cdot G = X$. Therefore x is given by $f(0)$. Note that 'strong' special soundness is achieved trivially as there is a unique z that satisfies any challenge e.

□

Efficiency. A single instance of the strongly r-special sound Schnorr is equivalent in bandwidth, proving, and verification cost to r copies of the regular 2-special sound Schnorr Sigma protocol. However each new prover response requires a factor of r more $\mathbb{Z}_q$ (scalar) multiplications to compute than a single copy of the regular 2-special sound Schnorr Sigma protocol. Our analysis, however, focuses on minimizing the number of hash queries to the random oracle.

Appendix B

Auxiliary Material for Stateless Deterministic Signing

B.1 UC Commitments

High level idea. As a helper for this protocol we first embed $\mathsf{Gen\text{-}vk}, \mathsf{Gen\text{-}ck}$ in a 'setup' functionality $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$, which establishes for the committer and receiver their respective keys (a corrupt party may supply the randomness ρ^S/ρ^R to $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$). The protocol itself consists of simply having S run Commit to commit to a message, and R verify openings with $\mathsf{DecomVrfy}$. The simulator for this protocol runs $\mathsf{Gen\text{-}ek}$ or $\mathsf{Gen\text{-}td}$ while acting on behalf of $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$ during setup. Subsequently the simulator uses the resulting keys ek or td to extract the message from commitments produced by a corrupt sender via Ext, or to equivocate messages to a corrupt receiver via $\mathcal{S}_{\mathsf{Com},\mathsf{R}^*}$ respectively.

We begin by giving the exact description of $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$ below:

Functionality B.1.1. $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$. Commitment Setup

This is a helper functionality for the UC-secure commitment protocol π_{Com}, run between a pair of parties S, R (one of whom may be corrupt). Given algorithms $\mathsf{Gen\text{-}ck}, \mathsf{Gen\text{-}vk}$ for commitment scheme $\mathcal{C}$, this functionality simply runs them to distribute the correct keys to S and R. Let $|\rho^S|$ and $|\rho^R|$ be the size of the random tapes required by $\mathsf{Gen\text{-}ck}$ and $\mathsf{Gen\text{-}vk}$ respectively. All messages are adversarially delayed.

Setup: Upon receiving (`init`) from both parties, do the following:

1. If neither party is corrupt, sample $\rho^S \leftarrow \{0,1\}^{|\rho^S|}$ and $\rho^R \leftarrow \{0,1\}^{|\rho^R|}$.
2. If S is corrupt, wait for $(\texttt{ck-rand}, \rho^S \in \{0,1\}^{|\rho^S|})$ from S and sample $\rho^R \leftarrow \{0,1\}^{|\rho^R|}$.

If R is corrupt, wait for $(\mathtt{vk\text{-}rand}, \rho^R \in \{0,1\}^{|\rho^R|})$ from R and sample $\rho^S \leftarrow \{0,1\}^{|\rho^S|}$.

3. Compute $\mathsf{ck} = \mathsf{Gen\text{-}ck}(1^\kappa; \rho^S)$ and $\mathsf{vk} = \mathsf{Gen\text{-}vk}(1^\kappa, \mathsf{ck}; \rho^R)$.

4. Send $(\mathtt{com\text{-}key}, \mathsf{ck})$ to S and $(\mathtt{ver\text{-}key}, \mathsf{vk})$ to R.

Accept no further commands.

With the helper functionality in place, we give the protocol for commitment π_{Com} below.

Protocol B.1.2. $\pi_{\mathsf{Com}}[\mathcal{C}]$**. Commitment**

This protocol is run between a sender S and a receiver R, and is parameterized by a commitment scheme $\mathcal{C}$. This protocol makes use of the ideal oracle $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$.

Setup: Run once:

1. S and R send $(\mathtt{init})$ to $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$

2. S receives $(\mathtt{com\text{-}key}, \mathsf{ck})$ and R receives $(\mathtt{ver\text{-}key}, \mathsf{vk})$ from $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$

Commit: With common input ind and private input m:
S computes $\mathsf{C}, \delta = \mathsf{Commit}(\mathsf{ck}, \mathsf{ind}, m)$ and sends C to R

Decommit: With common input ind:

1. S sends m, δ to R

2. R validates $\mathsf{DecomVrfy}(\mathsf{vk}, \mathsf{ind}, \mathsf{C}, \delta, m) \stackrel{?}{=} 1$

Equally important is to define the simulator for the above protocol.

Simulator B.1.3. $\mathcal{S}_{\mathsf{Com}}[\mathcal{C}]$**. Simulator for Protocol** π_{Com}

The simulator is parameterized by a commitment scheme $\mathcal{C}$, and acts on behalf of the ideal oracle $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$.

Corrupt sender: S^*

1. **Setup**:

   (a) Receive $(\mathtt{init})$ and $(\mathtt{ck\text{-}rand}, \rho^S)$ from S^* on behalf of $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$

   (b) Compute $\mathsf{ck} = \mathsf{Gen\text{-}ck}(1^\kappa; \rho^S)$ and $\mathsf{ek} = \mathsf{Gen\text{-}ek}(\mathsf{ck})$

   (c) Sample $\rho^R \leftarrow \{0,1\}^{|\rho^R|}$ and set $\mathsf{vk} = \mathsf{Gen\text{-}vk}(1^\kappa; \rho^R)$

(d) Send $(\texttt{com-key}, \mathsf{ck})$ to S^* on behalf of $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$

2. **Commit**: With common input ind:

   (a) Receive C from S^* on behalf of R

   (b) Compute $m = \mathsf{Ext}(\mathsf{ek}, \mathsf{ind}, \mathsf{C})$ and send $(\texttt{commit}, \mathsf{ind}, m)$ to $\mathcal{F}_{\mathsf{Com}}$

3. **Decommit**: With common input ind:

   (a) Receive m, δ on behalf of R

   (b) Compute $b = \mathsf{DecomVrfy}(\mathsf{vk}, \mathsf{ind}, \mathsf{C}, \delta, m)$ and send $(\texttt{reveal}, b)$ to $\mathcal{F}_{\mathsf{Com}}$

Corrupt receiver R^*: run $\mathcal{S}_{\mathsf{Com},\mathsf{R}^*}$ as necessary.

B.2 Committed OT Setup

We first define a functionality $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{OT}}$.

Functionality B.2.1. $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{OT}}$. All-but-one OT

This functionality allows a sender S to send ℓ messages and a receiver R to obtain all but one of them, while keeping S oblivious to which one was omitted. All outgoing messages are adversarially delayed.

Choose: Upon receiving $(\texttt{choose-all-but}, \mathsf{c})$ from R, and if $\mathsf{c} \in [\ell]$ and no such message was previously received, store $(\texttt{chosen}, \mathsf{c})$ in memory and send $(\texttt{chosen})$ to S.

Transfer: Upon receiving $(\texttt{transfer}, \{k_i\}_{i\in[\ell]})$ from S, if $(\texttt{chosen}, \mathsf{c})$ exists in memory then send $(\texttt{messages}, \{k_i\}_{i\in[\ell]\setminus\mathsf{c}})$ to R.

The functionality $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{OT}}$ can be realized assuming hardness of the Computational Diffie-Hellman problem in the same curve as the signature (in the random oracle model), by a simple adaptation of the classic Bellare-Micali OT protocol [BM90].

We now describe how to realize the all-but-one Oblivious Transfer functionality $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{OT}}$. The functionality $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{OT}}$ can be realized assuming hardness of the Computational Diffie-Hellman problem in the same curve as the signature (in the random oracle model), by a simple adaptation of the classic Bellare-Micali OT protocol [BM90], which we briefly recall: S samples $a \leftarrow \mathbb{Z}_q$, and sends $A = a \cdot G$ to R. Then, R samples $\ell - 1$ scalars indexed by all but c, as $(b_i)_{i\in[\ell]\setminus\mathsf{c}} \leftarrow \mathbb{Z}_q^{\ell-1}$, and sets $B_i = b_i \cdot G$ for each $i \in [\ell] \setminus \mathsf{c}$, and $B_{\mathsf{c}} = A - \sum_{i\in[\ell]\setminus\mathsf{c}} B_i$. R sends $(B_i)_{i\in[\ell]}$ to S, who verifies that $\sum_{i\in[\ell]} B_i = A$, and uses B_i as a public key to encrypt

message k_i. Observe that the receiver will be able to decrypt all messages except the one encrypted with B_{c}. In order to make this UC secure, the sender must prove knowledge of discrete log of A, and the receiver must prove knowledge of all-but-one discrete logs among $(B_i)_{i\in[\ell]}$, which is done via $\mathcal{F}_{\mathsf{.ZK}}^{\mathsf{R}_{DL}}$

The COT helper functionality is formally defined as follows:

Functionality B.2.2. $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}[\mathcal{C}]$**. Committed OT Setup**

This is a helper functionality for the committed OT protocol π_{COT}, run between a pair of parties S, R (one of whom may be corrupt). Let $|\rho^S|$ and $|\rho^R|$ be the size of the random tapes required by Gen-ck and Gen-vk respectively. All messages are adversarially delayed.

Setup: Upon receiving $(sid, \texttt{init})$ from both parties, do the following:

1. If neither party is corrupt, sample $\rho_0^S, \rho_1^S \leftarrow \{0,1\}^{2|\rho^S|}$ and $\rho_0^R, \rho_1^R \leftarrow \{0,1\}^{2|\rho^R|}$.
2. If S is corrupt, wait for $(sid, \texttt{ck-rand}, \rho_0^S, \rho_1^S \in \{0,1\})^{2|\rho^S|}$ from S and sample $\rho_0^R, \rho_1^R \leftarrow \{0,1\}^{2|\rho^R|}$. If R is corrupt, wait for $(sid, \texttt{vk-rand}, \rho_0^R, \rho_1^R \in \{0,1\})^{2|\rho^R|}$ from R and sample $\rho_0^S, \rho_1^S \leftarrow \{0,1\}^{2|\rho^S|}$.
3. Set commitment keys $\mathsf{ck}_0 = \mathsf{Gen\text{-}ck}(1^\kappa; \rho_0^S)$ and $\mathsf{ck}_1 = \mathsf{Gen\text{-}ck}(1^\kappa; \rho_1^S)$
4. Set verification keys $\mathsf{vk}_0 = \mathsf{Gen\text{-}vk}(\mathsf{ck}_0; \rho_0^R)$ and $\mathsf{vk}_1 = \mathsf{Gen\text{-}vk}(\mathsf{ck}_1; \rho_1^R)$
5. Compute extraction keys $\mathsf{ek}_0 = \mathsf{Gen\text{-}ek}(\mathsf{ck}_0)$ and $\mathsf{ek}_1 = \mathsf{Gen\text{-}ek}(\mathsf{ck}_1)$
6. Upon receiving $(sid, \texttt{choose}, b \in \{0,1\})$ from R, send $(sid, \texttt{keys}, \mathsf{ek}_b, \mathsf{vk}_0, \mathsf{vk}_1)$ to R, and $(\mathsf{ck\text{-}keys}, \mathsf{ck}_0, \mathsf{ck}_1)$ to S.

Accept no further commands.

We now give the exact protocol to realize the setup functionality $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$. Intuitively, the idea is for the sender to supply $A \in \mathbb{G}$, which the receiver splits into an additive sharing C_0, C_1 so that only one of the corresponding discrete logarithms (specifically that of C_b) is known. Following this, for each $j \in [\mathsf{r}]$, R further splits C_0, C_1 into ℓ additive shares each. Since R knows the discrete log of C_b, is also knows the discrete logs of every additive share. However since R does not know the discrete log of C_{1-b}, it will be missing the discrete log of exactly one of its additive shares.

Protocol B.2.3. $\pi_{\mathsf{COT}}^{\mathsf{setup}}$**. Setup protocol tailored to** $\mathcal{C}$

This protocol is run between a sender S and a receiver R. Parameters of the scheme are κ, and curve group $(\mathbb{G}, G, q)$. This protocol makes use of the ideal oracles $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{ZK}}^{\mathsf{R}_{DL}}$, $\mathcal{F}_{\mathsf{ZK}}^{\mathsf{R}_{DL}}$, and a random oracle RO.

Setup: P has no private input, and V has private input $b \in \{0,1\}$.

1. P samples $\mathsf{ck}_0 \leftarrow \{k_{0,j,l} \in \{0,1\}^\kappa\}_{j\in[\mathsf{r}],l\in[\ell]}$ and $\mathsf{ck}_1 \leftarrow \{k_{1,j,l} \in \{0,1\}^\kappa\}_{j\in[\mathsf{r}],l\in[\ell]}$
2. V samples $(i_j)_{j\in[\mathsf{r}]} \leftarrow [\ell]^{\mathsf{r}}$
3. P samples the OT sender's key, $a \leftarrow \mathbb{Z}_q$ and sends $A = a \cdot G$ to V and $(\texttt{prove}, a, A)$ to $\mathcal{F}_{\mathsf{ZK}}^{\mathsf{R}_{DL}}$
4. V waits for $(\texttt{proven}, A)$ from $\mathcal{F}_{\mathsf{ZK}}^{\mathsf{R}_{DL}}$
5. V samples $c_b \leftarrow \mathbb{Z}_q$ and computes $C_b = c_b \cdot G$, and $c_{1-b} = A - C$
6. V does the following, for each $j \in [\mathsf{r}]$:
   (a) Sample $(c_{b,j,l})_{l\in[\ell]} \leftarrow \mathbb{Z}_q^\ell$ such that $\sum_{l\in[\ell]} c_{b,j,l} = c_b$, and set $C_{b,j,l} = c_{b,j,l} \cdot G$ for each $l \in [\ell]$
   (b) Derive all keys for instance b as $k_{b,j,l} = \mathsf{RO}(c_{b,j,l} \cdot A)$ for each $l \in [\ell]$
   (c) Sample $(c_{1-b,j,l})_{l\in[\ell]\setminus i_j} \leftarrow \mathbb{Z}_q^{\ell-1}$, and set $C_{1-b,j,l} = c_{1-b,j,l} \cdot G$ for each $l \in [\ell] \setminus i_j$
   (d) Derive all keys but one for instance $1-b$ as $k_{1-b,j,l} = \mathsf{RO}(c_{1-b,j,l} \cdot A)$ for each $l \in [\ell] \setminus i_j$
   (e) Set the remaining $C_{1-b,j,l} = C_{1-b} - \sum_{l\in[\ell]\setminus i_j} C_{1-b,j,l}$
   (f) Assemble length $2\ell - 1$ vector that has every $c_{\cdot,j,\cdot}$ value except the one indexed by $1-b, j, i_j$: $\mathbf{c}_j = \{c_{b,j,l}\}_{l\in[\ell]} \cup \{c_{1-b,j,l}\}_{l\in[\ell]\setminus i_j}$
   (g) Send $(\texttt{prove-all-but-one}, i_j, \mathbf{c}_j, (C_{0,j,l}, C_{1,j,l})_{l\in[\ell]})$ to $\mathcal{F}_{\binom{2\ell}{2\ell-1}\mathsf{ZK}}^{\mathsf{R}_{DL}}$
7. V sends $C_0, C_1, (C_{0,j,l}, C_{1,j,l})_{j\in[\mathsf{r}],l\in[\ell]}$ to P
8. P verifies that $C_0 + C_1 = A$, and does the following for each $j \in [\mathsf{r}]$:
   (a) Wait to receive $(\texttt{proven-all-but-one}, (C_{0,j,l}, C_{1,j,l})_{l\in[\ell]})$ from $\mathcal{F}_{\binom{2\ell}{2\ell-1}\mathsf{ZK}}^{\mathsf{R}_{DL}}$
   (b) Verify that $\sum_{l\in[\ell]} C_{0,j,l} = C_0$ and $\sum_{l\in[\ell]} C_{1,j,l} = C_1$
   (c) For each $l \in [\ell]$: Compute $k_{0,j,l} = \mathsf{RO}(a \cdot C_{0,j,l})$ and $k_{1,j,l} = \mathsf{RO}(a \cdot C_{1,j,l})$

9. P outputs $\mathsf{ck} = \{k_{0,j,l}, k_{1,j,l}\}_{j\in[\mathsf{r}],l\in[\ell]}$
   V outputs $\mathsf{vk} = \{k_{b,j,l}\}_{j\in[\mathsf{r}],l\in[\ell]}\} \cup \{(k_{1-b,j,l})_{l\in[\ell]\setminus i_j}\}_{j\in[\mathsf{r}]}$

Theorem B.2.4. *Assuming that the Computational Diffie-Hellman problem is hard in $\mathbb{G}$, protocol $\pi_{\mathsf{COT}}^{\mathsf{setup}}$ UC-realizes $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$ in the $\mathcal{F}_{\binom{2\ell}{2\ell-1}\mathsf{ZK}}^{\mathsf{R}_{DL}}$,$\mathcal{F}_{\mathsf{ZK}}^{\mathsf{R}_{DL}}$-hybrid local random oracle model.*

B.3 Garbling Scheme Definitions

We recall here the formal definitions of garbling schemes.

Definition B.3.1. *(Correctness) A garbling scheme $\mathcal{G}$ is* correct *if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$ and inputs $x \in \{0,1\}^n$, the following probability is negligible in κ:*

$$\Pr\left(\mathsf{De}(\mathsf{Ev}(\tilde{C}, \mathsf{En}(\mathsf{en}, x)), \mathsf{de}) \neq C(x) : (\tilde{C}, \mathsf{en}, \mathsf{de}) \leftarrow \mathsf{Gb}(1^\kappa, C)\right).$$

Definition B.3.2. *(Authenticity) A garbling scheme $\mathcal{G}$ is* authentic *if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$, inputs $x \in \{0,1\}^n$, and all probabilistic polynomial-time adversaries $\mathcal{A}$, the following probability is negligible in κ:*

$$\Pr\left(\begin{matrix}\widehat{Y} \neq \mathsf{Ev}(\tilde{C}, X) \\ \wedge \mathsf{De}(\widehat{Y}, \mathsf{de}) \neq \bot\end{matrix} : \begin{matrix}X = \mathsf{En}(x, \mathsf{en}),\ (\tilde{C}, \mathsf{en}, \mathsf{de}) \leftarrow \mathsf{Gb}(1^\kappa, C) \\ \widehat{Y} \leftarrow \mathcal{A}(C, x, \tilde{C}, X)\end{matrix}\right).$$

Definition B.3.3. *(Verifiability) A garbling scheme $\mathcal{G}$ is* verifiable *if for all input lengths $n \leq \mathsf{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$, inputs $x \in \{0,1\}^n$, and PPT adversaries $\mathcal{A}$, the following probability is negligible in κ:*

$$\Pr\left(\mathsf{De}\left(\mathsf{Ev}(\tilde{C}, \mathsf{En}(x, \mathsf{en})), \mathsf{de}\right) \neq C(x) \ : \begin{matrix}(\tilde{C}, \mathsf{en}) \leftarrow \mathcal{A}(1^\kappa, C) \\ \mathsf{Ve}\left(C, \tilde{C}, \mathsf{en}\right) = \mathsf{de}\end{matrix}\right)$$

For completeness, we also require the following property of a verifiable garbling scheme:

$$\forall \left(\tilde{C}, \mathsf{en}, \mathsf{de}\right) \leftarrow \mathsf{Gb}\left(1^\kappa, C\right),\ \mathsf{Ve}\left(C, \tilde{C}, \mathsf{en}, \mathsf{de}\right) = 1$$

Appendix C

Auxiliary Material for Proactive Threshold Signing

C.1 Threshold Schnorr Signing

We recall below the folklore instantiation of threshold Schnorr signatures.

Protocol C.1.1. $\pi^{\mathsf{DKG}}_{\mathsf{Setup}}$**. Distributed Key Generation for Schnorr**

Parameters: Elliptic Curve Group $(\mathbb{G}, G, q)$
Parties: P_i for $i \in [n]$
Ideal Oracles: $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$
Outputs:

- **Common**: Public key $\mathsf{pk} \in \mathbb{G}$
- **Private**: Secret key share sk_i

---

1. Each party P_i samples a random degree-1 polynomial f_i over $\mathbb{Z}_q$
2. For all pairs of parties P_i and P_j, P_i sends $f_i(j)$ to P_j and receives $f_j(i)$ in return.
3. Each party P_i computes its point
$$f(i) := \sum_{j \in [1,n]} f_j(i)$$
4. Each P_i computes
$$T_i := f(i) \cdot G$$

and sends $(\texttt{com-proof}, \mathsf{id}_i^{\text{com-zk}}, f(i), T_i)$ to $\mathcal{F}_{\text{Com-ZK}}^{\mathsf{R}_{DL}}$, using a fresh, unique value for $\mathsf{id}_i^{\text{com-zk}}$.

5. Upon being notified of all other parties' commitments, each party P_i releases its proof by sending $(\texttt{decom-proof}, \mathsf{id}_i^{\text{com-zk}})$ to $\mathcal{F}_{\text{Com-ZK}}^{\mathsf{R}_{DL}}$.

6. Each party P_i receives $(\texttt{accept}, \mathsf{id}_j^{\text{com-zk}}, T_j)$ from $\mathcal{F}_{\text{Com-ZK}}^{\mathsf{R}_{DL}}$ for each $j \in [1, n] \setminus \{i\}$ if P_j's proof of knowledge is valid. P_i aborts if it receives $(\texttt{fail}, \mathsf{id}_j^{\text{com-zk}})$ instead for any proof, or if there exists an index $x \in [3, n]$ such that

$$\lambda_1^2(x) \cdot T_1 + \lambda_2^1(x) \cdot T_2 \neq T_x$$

7. The parties compute the shared public key as

$$\mathsf{pk} := \lambda_1^2(0) \cdot T_1 + \lambda_2^1(0) \cdot T_2$$

The above protocol is a reproduction of the distributed key generation protocol of Pedersen [**?**], adjusted for context.

Protocol C.1.2. $\pi_{\text{Schnorr}}^{\mathsf{R}}(\mathsf{pk}, \mathsf{sk}_b, 1-b, m)$**. Schnorr Signing—Nonce Generation**

Parameters: Elliptic Curve Group $(\mathbb{G}, G, q)$
Parties: P_b, P_{1-b} for $b, 1-b \in [n]$
Ideal Oracles: $\mathcal{F}_{\text{Com-ZK}}^{\mathsf{R}_{DL}}$
Inputs:

- **Common**: Message to be signed $m \in \{0,1\}^*$, public key $\mathsf{pk} \in \mathbb{G}$, each party's share in the exponent $\mathsf{pk}_b = \lambda_b^{1-b}(0) \cdot F(b)$ where F is the polynomial over $\mathbb{G}$ passing through $(0, \mathsf{pk})$ and $(b, f(b) \cdot G)$
- **Private**: Each party P_b has private input $\mathsf{sk}_b = \lambda_b^{1-b}(0) \cdot f(b) \in \mathbb{Z}_q$

Outputs:

- **Common**: Signing nonce $R \in \mathbb{G}$
- **Private**: Each party P_b has private output $\mathsf{state}_b \in \mathbb{Z}_q$

---

1. Include all inputs in state_n
2. Sample $k_b \leftarrow \mathbb{Z}_q$ and send $(\texttt{commit}, k_b, R_b = k_b \cdot G)$ to $\mathcal{F}_{\text{Com-ZK}}^{\mathsf{R}_{DL}}$ with fresh identifier $\mathsf{id}_b^{\text{com-zk}}$

3. Upon receiving $(\texttt{committed}, 1-b, \mathsf{id}^{\mathsf{com}}_{1-b})$ from $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$, instruct $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$ to release R_b

4. Upon receiving $(\texttt{decommitted}, 1-b, \mathsf{id}^{\mathsf{com}}_{1-b}, R_{1-b})$ from $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$ if $R_{1-b} \in \mathbb{G}$ then compute

$$R = R_b + R_{1-b}$$

5. Include k_b in state_b

6. Output state_b, R

Protocol C.1.3. $\pi^{\sigma}_{\mathbf{Schnorr}}(\mathsf{state}_b)$**. Schnorr Signing—Completion**

Parameters: Elliptic Curve Group $(\mathbb{G}, G, q)$
Parties: P_b, P_{1-b} for $b, 1-b \in [n]$
Ideal Oracles: $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$
Inputs: (Encoded in state_b)

- **Common**: Message to be signed $m \in \{0,1\}^*$, public key $\mathsf{pk} \in \mathbb{G}$
- **Private**: Each party P_b has private input $\mathsf{sk}_b = \lambda_b^{1-b}(0) \cdot f(b) \in \mathbb{Z}_q$

---

1. Parse $k_b, m, \mathsf{sk}_b \leftarrow \mathsf{state}_b$

2. Compute

$$\sigma_b = H(R||m) \cdot \mathsf{sk}_b + k_b$$

and send σ_b to P_{1-b}

3. Upon receiving $\sigma_{1-b} \in \mathbb{Z}_q$ from P_{1-b} compute

$$\sigma = \sigma_b + \sigma_{1-b}$$

and if (σ, R) is a valid Schnorr signature under public key pk then output σ

By the linearity of the Schnorr signing equation, it is easy to verify correctness as

$$\begin{aligned}
\sigma &= \sigma_b + \sigma_{1-b} \\
&= \left(H(R||m) \cdot \lambda_b^{1-b}(0) \cdot \mathsf{sk}_b + k_b\right) \\
&\quad + \left(H(R||m) \cdot \lambda_{1-b}^{b}(0) \cdot \mathsf{sk}_{1-b} + k_{1-b}\right) \\
&= H(R||m) \cdot \left(\lambda_b^{1-b}(0) \cdot \mathsf{sk}_b + \lambda_{1-b}^{b}(0) \cdot \mathsf{sk}_{1-b}\right) \\
&\quad + (k_b + k_{1-b}) \\
&= H(R||m) \cdot \mathsf{sk} + k
\end{aligned}$$

Theorem C.1.4. *(Informal) The protocol* $(\pi^{\mathsf{DKG}}_{\mathsf{Setup}}, \pi^{\mathsf{R}}_{Schnorr}, \pi^{\sigma}_{Schnorr})$ *UC-realizes* $\mathcal{F}^{n,2}_{\mathsf{Sign}}$ *for* $\mathsf{Sign} = \mathsf{Sign}_{Schnorr}$ *in the* $\mathcal{F}_{\mathsf{Com}}, \mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$*-hybrid model.*

The simulation strategy is straightforward: $\mathcal{S}^{\mathsf{R}}_{\text{Schnorr}}$ upon receiving R from the functionality sends $R_{1-b} = R - R_b$ to P_b (on behalf of $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$). The simulator $\mathcal{S}^{\sigma}_{\text{Schnorr}}$ upon receiving σ from the functionality sends $\sigma_{1-b} = \sigma - \sigma_b$ to P_b on behalf of P_{1-b}. Note here that σ_b is computed by the simulator as instructed by Step 2 of $\pi^{\sigma}_{\text{Schnorr}}$ using the value k_b received on behalf of $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$ in Step 2 of $\pi^{\mathsf{R}}_{\text{Schnorr}}$.