Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions

François Garillot
francois@garillot.net
Novi/Facebook

Yashvanth Kondi*
ykondi@ccs.neu.edu
Northeastern University

Payman Mohassel
paymanm@fb.com
Facebook

Valeria Nikolaenko
valerini@fb.com
Novi/Facebook

Abstract

Schnorr's signature scheme permits an elegant threshold signing protocol due to its linear signing equation. However each new signature consumes fresh randomness, which can be a major attack vector in practice. Sources of randomness in deployments are frequently either unreliable, or require *state continuity*, i.e. reliable fresh state resilient to rollbacks. State continuity is a notoriously difficult guarantee to achieve in practice, due to system crashes caused by software errors, malicious actors, or power supply interruptions (Parno et al., S&P '11). This is a non-issue for Schnorr variants such as EdDSA, which is specified to derive nonces deterministically as a function of the message and the secret key. However, it is challenging to translate these benefits to the threshold setting, specifically to construct a threshold Schnorr scheme where signing neither requires parties to consume fresh randomness nor update long-term secret state.

In this work, we construct a dishonest majority threshold Schnorr protocol that enables such stateless deterministic nonce derivation using standardized block ciphers. Our core technical ingredients are new tools for the zero-knowledge from garbled circuits (ZKGC) paradigm to aid in verifying correct nonce derivation:

- A mechanism based on UC Commitments that allows a prover to commit once to a witness, and prove an unbounded number of statements online with only cheap symmetric key operations.
- A garbling gadget to translate intermediate garbled circuit wire labels to arithmetic encodings.

Our scheme prioritizes computation cost, with each proof requiring only a small constant number of exponentiations.

*Part of this work was done during an internship at Novi/Facebook.

1 Introduction

The Schnorr signature scheme [Sch91] is a simple discrete logarithm based construction where the public key is of the form $X = x \cdot G$, and to sign a message m the signer provides $R = r \cdot G$ and the linear combination $\sigma = xe + r$ where e is derived by hashing X, R, m (or just R, m as in the original specification). This relation can easily be verified in the exponent as $\sigma \cdot G = e \cdot X + R$. Assuming an ideal hash function (modelled as a random oracle) unforgeability is rooted in the hardness of the discrete logarithm problem. The signatures themselves embody a zero-knowledge property: assuming that the r values are uniformly chosen, the σ values reveal *no information* about x.

1.1 Practical Concerns: Determinism and Statelessness

The fact that r values are chosen uniformly upon every invocation 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].

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 work, 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.

1.2 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.

1.3 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.

1.4 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 Half-Gates [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.

1.4.1 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:

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

> 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 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 summarize below in Table 1.1.

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 1.1 suggests that our technique for preprocessing Committed OT can perform input encoding an order of magnitude faster than using the fastest plain OT.

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 1.1: 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.

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.

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.

1.4.2 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 below in Table 1.2.

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 1.2: 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.

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].

Intuition. The gadget is inspired by the Oblivious Linear Evaluation technique of Gilboa [Gil99]. 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 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.

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 1.2. 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.

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 distributed 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 $\mathcal{F}_{\mathsf{F}\cdot\mathbb{G}}$ later in this paper.

3.1 What existing proof technologies suit our task?

As per our desiderata that we set in Section 1.3, 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. $O(|q|)$ asymptotically 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 1.4.

4 Organization

We first establish the technical background in Section 5. We then expand on our solutions to the gaps that we identified in Section 1.4: Section 6 details the garbling gadget for exponentiation, and Section 7 elaborates on how to construct Committed OT from UC Commitments. Section 8 shows how to combine these ideas to build a nonce verifying mechanism, and finally Section 9 constructs an n-party protocol resilient to $n-1$ corruptions based on this mechanism.

5 Preliminaries

Security Model. We construct and prove our protocols secure in the Universal Composability framework of Canetti [Can01]. We assume synchronous networks, with well-defined upper bounds on adversarial message delay.
Standard Helper Functionalities. We define choose all-but-one OT $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{OT}}$ in Appendix B, and some other useful standard functionalities in Appendix C.

5.1 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.

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**[2]: 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

[2]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.

any PPT adversary $\mathcal{A}(\tilde{C}, \tilde{X})$ to output $\hat{Z}$ such that $\mathsf{De}(\mathsf{de}, \hat{Z}) = \hat{y}$.

- **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 D.

5.1.1 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 7.

5.1.2 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.

5.1.3 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.

Subsequently 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 $\tilde{Y}$.

6 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 6.1. $\mathcal{G}_{\mathsf{exp}}$. 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 6.2 (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 6.3. *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 $\hat{Z}$:

$$\begin{aligned} &\Pr\left[\hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}) : (\tilde{C}, \mathsf{en}, \mathsf{de}) \leftarrow \mathsf{Gb}(f_{\mathbf{u}}), \tilde{X} \leftarrow \mathsf{En}(\mathsf{en}, \mathbf{x})\right] \\ &= \Pr\left[\hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}) : (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_1(f_{\mathbf{u}}, \mathbf{x})\right] \end{aligned} \tag{1}$$

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 $\hat{Z} \in \mathbb{G}$:

$$\left| \begin{aligned} &\Pr[\hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}) : (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_2(f_{\mathbf{u}}, \mathbf{x})] \\ &- \Pr[\hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}) : (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_1(f_{\mathbf{u}}, \mathbf{x})] \end{aligned} \right| \leq \mathsf{negl}(\kappa) \tag{2}$$

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,

$$\begin{aligned} &\Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = Y : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_2(f_{\mathbf{u}}, \mathbf{x})\right] \\ &= \Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = Y : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_3(f_{\mathbf{u}}, \mathbf{x})\right] \end{aligned} \tag{3}$$

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:

$$\begin{aligned} &\Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = Y : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_3(f_{\mathbf{u}}, \mathbf{x})\right] \\ &= \Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = Y : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_4(f_{\mathbf{u}}, \mathbf{x})\right] \end{aligned} \tag{4}$$

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 $\mathcal{A}$,

$$\Pr\left[\mathsf{De}(\mathsf{de}, \hat{Z}) = \hat{Y} : \hat{Z} \leftarrow \mathcal{A}(\tilde{C}, \tilde{X}),\ (\tilde{C}, \tilde{X}, \mathsf{de}) \leftarrow \mathcal{H}_4(f_{\mathbf{u}}, \mathbf{x})\right] = 1/(q-1) \tag{5}$$

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

Combining equations 1-5 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.

□

7 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 7.1. $\mathcal{F}^*_{\mathsf{COT}}$**. 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 $(sid, \texttt{message}, \mathsf{ind}, m_b)$ to R.

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

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 7.2. *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 $\mathcal{C}$.

7.0.1 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 7.3. $\mathcal{C}$**. 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 7.4.

Theorem 7.4. *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.

7.1 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. 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.

Protocol 7.5. $\pi_{\mathsf{COT}}[\mathcal{C}]$**. 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 7.6. *Assuming $\mathcal{C}$ is a preprocessable UC commitment (Def. 7.2), 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.

7.1.1 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 $O(\ell r \kappa / \log \kappa)$ 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 7.0.1, 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.

8 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 8.1. $\mathcal{F}_{\mathsf{F},\mathbb{G}}$**. 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 8.2. $\pi_{\mathsf{F}\cdot\mathbb{G}}$**. 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, $\rho_{\mathcal{G}} = \mathsf{RO}(k_V, \mathsf{ind}, \texttt{Gb})$
   (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 $(i, \texttt{transfer}, \mathsf{ind}, \mathsf{key}, \mathsf{en}_{i,0}, \mathsf{en}_{i,1})$ to $\mathcal{F}^*_{\mathsf{COT}}$

3. Upon receiving $(\tilde{C}, \zeta)$ from V and $(i, \texttt{message}, \mathsf{ind}, \mathsf{en}_{i,k_i})$ 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 8.3. *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.

□

8.1 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.

8.2 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 7.1.1, 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.

9 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 $\mathcal{F}_{\mathsf{F}.\mathbb{G}}$ 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 9.1. $\mathcal{F}_{n,\mathsf{Sign}}$**. 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 subsequent requests.

1. Wait to receive (`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 9.2. $\pi_{n,\mathsf{Sign}}$. **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 $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$
6. Upon receiving $(\texttt{committed}, j)$ from $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$ for each $j \in [n] \setminus i$, send $(\texttt{reveal})$ to $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$
7. Wait to receive $(\texttt{revealed}, \mathsf{pk}_j)$ from $\mathcal{F}^{\mathsf{R}_{DL}}_{\mathsf{Com-ZK}}$ 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 9.3. *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. □

9.1 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 8.2. This cost is multiplied by n for an n party instantiation.

10 Acknowledgements

The authors would like to thank Jack Doerner, Tim Ruffing, and the anonymous reviewers for their helpful comments and feedback.

Bibliography

[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 *ACM CCS 2018*. ACM Press, 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*, 2018.

[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 *EUROCRYPT 2019, Part I*, 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*, 2012.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020*, 2020.

[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 *PKC 2019, Part I*, 2019.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM CCS 2012*, 2012.

[BM90] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *CRYPTO'89*, 1990.

[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 hasheddsa: 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*, 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.

[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 *PKC 2015*, 2015.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO'94*, 1994.

[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 *ACM CCS 2017*, 2017.

[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 *CRYPTO 2016, Part III*, 2016.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In *ACM CCS 2014*, 2014.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In *TCC 2012*, 2012.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *LATINCRYPT 2015*, 2015.

[DDGN14] Ivan Damgård, Bernardo Machado David, Irene Giacomelli, and Jesper Buus Nielsen. Compact VSS and efficient homomorphic UC commitments. In *ASIACRYPT 2014, Part II*, 2014.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *CRYPTO 2002*, 2002.

[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.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO 2005*, 2005.

[FJNT16] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. On the complexity of additively homomorphic UC commitments. In *TCC 2016-A, Part I*, 2016.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *EUROCRYPT 2015, Part II*, 2015.

[GIKW14] Juan A. Garay, Yuval Ishai, Ranjit Kumaresan, and Hoeteck Wee. On the complexity of UC commitments. In *EUROCRYPT 2014*, 2014.

[Gil99] Niv Gilboa. Two party RSA key generation. In *CRYPTO'99*, 1999.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 2007.

[GKPS18] Chaya Ganesh, Yashvanth Kondi, Arpita Patra, and Pratik Sarkar. Efficient adaptively secure zero-knowledge from garbled circuits. In *PKC 2018, Part II*, 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.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, 1987.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016, Part II*, 2016.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In *EUROCRYPT*, 2020.

[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 *ASIACRYPT 2017, Part I*, 2017.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *ACM CCS 2013*, 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.

[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.

[Kos] Ahmed Kosba. xJsnark.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT 2018, Part III*, 2018.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In *CRYPTO 2018, Part III*, 2018.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP 2008, Part II*, 2008.

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In *ACM CCS 2003*, 2003.

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In *EUROCRYPT 2011*, 2011.

[LPR19] Yehuda Lindell, Guy Peer, and Samuel Ranellucci. Unbound blockchain-crypto-mpc library. *White Paper*, 2019.

[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 *USENIX Security 2017*, 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 *SAC 1998*, 1999.

[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*, 1999.

[NKDM03] Antonio Nicolosi, Maxwell N. Krohn, Yevgeniy Dodis, and David Mazières. Proactive two-party signatures for user authentication. In *NDSS 2003*, 2003.

[NRSW20] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces. *ACM CCS 2020*, 2020.

[Ped91] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *EUROCRYPT'91*, 1991.

[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*, 2011.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *EUROCRYPT'96*, 1996.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 1991.

[Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. *CRYPTO 2020*, 2020.

[SP16] Raoul Strackx and Frank Piessens. Ariadne: A minimal approach to state continuity. In *USENIX Security 2016*, 2016.

[SS01] Douglas R. Stinson and Reto Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In *ACISP 01*, 2001.

[TTA18] Akira Takahashi, Mehdi Tibouchi, and Masayuki Abe. New Bleichenbacher records: Fault attacks on qDSA signatures. *IACR TCHES*, 2018.

[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.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT 2015, Part II*, 2015.

A UC Commitments

We first give the commitment functionality itself.

Functionality A.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.

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 A.2. $\mathcal{F}_{\mathsf{COT}}^{\mathsf{setup}}$**. 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 $(\texttt{init})$ 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 $(\texttt{vk-rand}, \rho^R \in \{0,1\}^{|\rho^R|})$ 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 $(\texttt{com-key}, \mathsf{ck})$ to S and $(\texttt{ver-key}, \mathsf{vk})$ to R.

Accept no further commands.

With the helper functionality in place, we give the protocol for commitment π_{Com} below.

Protocol A.3. $\pi_{\mathsf{Com}}[\mathcal{C}]$**. 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 $(\texttt{init})$ to $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}$
2. S receives $(\texttt{com-key}, \mathsf{ck})$ and R receives $(\texttt{ver-key}, \mathsf{vk})$ 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 A.4. $\mathcal{S}_{\mathsf{Com}}[\mathcal{C}]$. 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 $(\texttt{init})$ and $(\texttt{ck-rand}, \rho^S)$ 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 Committed OT Setup

We first define a functionality $\mathcal{F}_{\binom{\ell}{\ell-1}\mathsf{OT}}$.

Functionality B.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. $\mathcal{F}_{\mathsf{Com}}^{\mathsf{setup}}[\mathcal{C}]$**. 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 $(\texttt{ck-keys}, \mathsf{ck}_0, \mathsf{ck}_1)$ 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.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.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.*

We postpone the proof of the above theorem to the full version of this paper.

C Useful Functionalities

Functionality C.1. 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 C.2. Prove knowledge of 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 straight-line extractable sigma protocol to NIZK compiler in the random oracle model [Fis05].

D Garbling Scheme Definitions

We recall here the formal definitions of garbling schemes.

Definition D.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 D.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 D.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:

$$\forall \left(\tilde{C}, \mathsf{en}, \mathsf{de}\right) \leftarrow \mathsf{Gb}\left(1^\kappa, C\right),\ \mathsf{Ve}\left(C, \tilde{C}, \mathsf{en}, \mathsf{de}\right) = 1$$