Cryptography in the Multi-string Model

Jens Groth* Rafail Ostrovsky†

UCLA Computer Science Department
4732 Boelter Hall
Los Angeles, CA 90095-1596, USA
{jg,rafail}@cs.ucla.edu

Abstract

The common random string model permits the construction of cryptographic protocols that are provably impossible to realize in the standard model. In this model, a trusted party generates a random string and gives it to all parties in the protocol. However, the introduction of such a third party should set alarm bells going off: Who is this trusted party? Why should we trust that the string is random? Even if the string is uniformly random, how do we know it does not leak private information to the trusted party? The very point of doing cryptography in the first place is to prevent us from trusting the wrong people with our secrets.

In this paper, we propose the more realistic multi-string model. Instead of having one trusted authority, we have several authorities that generate random strings. We do not trust any single authority, we only assume a majority of them generate the random string honestly. We demonstrate the use of this model for two fundamental cryptographic taks. We define non-interactive zero-knowledge in the multi-string model and construct NIZK proofs in the multi-string model. We also consider multi-party computation and show that any functionality can be securely realized in the multi-string model.

Keywords: Common random string model, multi-string model, non-interactive zero-knowledge, multi-party computation.

*Supported by NSF ITR/Cybertrust grant No. 0456717.

†Supported in part by a gift from Teradata, Intel equipment grant, NSF Cybertrust grant No. 0430254, OKAWA research award, B. John Garrick Foundation and Xerox Innovation Group Award.

1 Introduction

THE PROBLEM. In the common random string model, a trusted party generates a uniformly random bit-string and makes it available to all parties. A generalization of this model is the common reference string (CRS) model, where the string may have a non-uniform distribution. Blum, Feldman and Micali [BFM88] introduced the CRS model to construct non-interactive zero-knowledge (NIZK) proofs. A relaxation of the plain model was needed, since only languages in BPP can have non-interactive or two-round NIZK proofs in the plain model, [GO94]. There are other examples of protocols that cannot be realized in the standard model but are possible in the CRS model, for instance universally composable (UC) commitment [CF01]. The CRS-model has therefore found wide-spread use in the field of cryptology.

Using the CRS-model to solve the tasks mentioned above in some sense just ignores a very real problem. It remains to specify where the CRS comes from. One solution is to have a trusted third party that generates the CRS, but this raises a trust-issue. It is very possible that the parties cannot find a party that they all trust. Would Apple trust a CRS generated by Microsoft? Would US government agencies be willing to use a CRS generated by their Russian counterparts?

Alternatively, the parties can generate the CRS themselves at the beginning of the protocol. If a majority is honest, they could for instance use multi-party computation to generate a CRS. However, this kind of setup makes the whole protocol much more complicated and requires them to have an initial round of interaction. They could also trust a group of parties to jointly generate a CRS, however, this leaves them with the task of finding a volunteer group to run a multi-party computation protocol whenever a CRS is needed. Other relaxations of the CRS-model found in the literature, such as the registered public key model that Barak et al.[BCNP04] use for multi-party computation also suffer from deficiencies. In the registered key model, parties can register correctly generated public keys, and these keys can be used for multi-party computation. However, now we need a trusted party to perform this verification of the keys.

THE MULTI-STRING MODEL. We propose the multi-string model as a solution to the above mentioned problems. In this model we have a number of authorities that assist the protocol execution by providing random strings. If a majority of these authorities are honest the protocol will be secure. There are two reasons that the multi-string model is attractive. First, the authorities play a minimal role in the protocol. They simply publish random strings, they do not need to perform any computation, be aware of each other or any other parties, or have any knowledge about the specifics of the protocol to be executed. This permits easy implementation, the parties wishing to execute a protocol can for instance simply download a set of random strings from agreed upon authorities on the internet. Second, the security of the protocols need to rely only on a majority of the authorities being honest at the time they created the strings. No matter how untrustworthy the other parties in your protocol are, you can trust the protocol if a majority of the authorities are honest. In other words, the honesty of a small group of parties can be magnified and used by any set of parties.

Now we have a new reasonable model for constructing secure protocols. The question remains, whether we can actually securely realize protocols in this model? We answer this question in the affirmative by defining and constructing non-interactive zero-knowledge proofs in the multi-string model and by securely realizing general multi-party computation in the multi-string model.

1.1 Non-interactive Zero-Knowledge

A zero-knowledge proof is a two-party protocol, where a prover tries to convince a verifier about the truth of some statement, typically membership of an NP-language. The proof should only convince the verifier if indeed the statement is true, however, at the same time the proof should reveal no extra information to the verifier other than the truth of the statement, in particular it should not reveal the NP-witness known to the prover. Interactive zero-knowledge proofs are known to exist in the standard model, however, as mentioned before non-interactive and 2-round zero-knowledge proofs only exist for trivial languages [GO94]. Instead, much research has gone into constructing non-interactive zero-knowledge proofs in the CRS-model.

We define multi-string NIZK proofs for NP-languages in Section 2. In this definition a proof is constructed using n common reference strings, which we imagine to be picked from a set of strings generated by some authorities. We further imagine that out of the n strings, a majority (or some threshold, see Section 2) of them have been honestly generated and no side-information has been stored about them. In that case we will have completeness, soundness and zero-knowledge defined as respectively the prover being able to convince the verifier if he knows a witness for the statement, the prover's inability to prove a false statement and the verifier's inability to learn anything else from the proof than the truth of the statement. We also consider more complex notions of zero-knowledge such as simulation-soundness, proofs of knowledge and simulation-sound extractability in the multi-string model.

We will construct multi-string NIZK proofs for any NP-language based on general cryptographic assumptions. This is a non-trivial task, since any of the common reference strings may be maliciously generated and leak information, so for instance the trivial solution of concatenating n NIZK proofs does not work.

We also construct very efficient multi-string NIZK proofs for circuit satisfiability, based on specific number theoretic assumption related to groups with bilinear maps. Using groups with a bilinear map is harder than one might expect at first glance, since all the common reference strings are generated independently and therefore we cannot assume the existence of a commonly agreed upon group. Nonetheless, we manage to construct such NIZK proofs and they are surprisingly efficient, a proof consists of $\mathcal{O}(n + |C|)$ group elements. Since in a typical setting, n will be much smaller than the size of the circuit this matches the most efficient known constructions for the single common reference string case by Groth, Ostrovsky and Sahai [GOS06b, GOS06a] where an NIZK proof consists of $\mathcal{O}(|C|)$ group elements.

1.2 Multi-party Computation

Canetti's UC framework [Can01] defines secure execution of a protocol under concurrent execution of arbitrary protocols. We refer the reader to Section 5.1 for an overview and to Canetti's paper for details, for now let us just say that the essence of the definition is to compare a protocol executing in the real world with an ideal process where a trusted party takes inputs from the parties and hands them their outputs. A protocol securely realizes the ideal functionality (the trusted party's program) if whatever the executing environment sees in the real life execution can be simulated on top of the ideal functionality.

It is known that in the plain model, any (well-formed) ideal functionality can be securely realized if a majority of the parties are honest. On the other hand, if a majority may be corrupt, there are certain functionalities that are provably impossible to realize. Relaxing the setting to the CRS-model, Canetti, et al. showed that any (well-formed) ideal functionality can be securely realized in the CRS-model, even against adversaries that can adaptively corrupt arbitrary parties and where parties are not assumed to be able to securely erase any of their data. However, it is an open question where this CRS should come from, since the parties provably cannot compute it themselves and it may be undesirable to trust one single authority to create a CRS, and risk the compromise of all your confidential data if the trust turns out to be unwarranted.

In this paper, we will show that any well-formed functionality can be securely realized in the multi-string model. This is a significant step forward, since even mutually distrustful parties may still agree on a set of authorities where they trust that some subset will be honest enough to generate good common reference strings. Also, there is now much less incentive for any given authority to cheat since to learn anything from the protocol it would need to risk cooperation with other authorities and face a higher risk of being caught. For instance the honest-but-curious system administrator who in the single authority setup might generate a common reference string that permitted decryption of the parties' secrets, no longer learns anything and therefore has less incentive to generate a fake common reference string.

2 Definitions

Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$ we call x the statement and w the witness. Let L be the NP-language consisting of statements in R.

A multi-string proof system for a relation R consists of probabilistic polynomial time algorithms K, P, V, which we will refer to as respectively the key generator, the prover and the verifier.

The key generation algorithm can be used to produce common reference strings σ. In the present paper, we can implement our protocols with a key generator that outputs a uniformly random string of polynomial length $\ell(k)$, however, for the sake of generality, we include a key generator in our definitions. Please note, the key generator takes only the security parameter as input, we do not assume that the key generator has any knowledge of the circumstances in which the common reference string is going to be used.

The prover takes as input $(t_c, t_s, t_z, \vec{\sigma}, x, w)$, where $\vec{\sigma}$ is a set on n common reference strings and $(x, w) \in R$, and produces a proof π. The verifier takes as input $(t_c, t_s, t_z, \vec{\sigma}, x, \pi)$ and outputs 1 if the proof is acceptable and 0 if rejecting the proof. We call (K, P, V) a (t_c, t_s, t_z, n)-NIZK proof system for R if it has the completeness, soundness and zero-knowledge properties described below. We remark that $(1, 1, 1, 1)$-NIZK proof systems correspond closely to the standard notion of NIZK proofs in the CRS-model.

(t_c, t_s, t_z, n)-COMPLETENESS. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\Pr\Big[S := \emptyset; (\vec{\sigma}, x, w) \leftarrow \mathcal{A}^K; \pi \leftarrow P(t_c, t_s, t_z, \vec{\sigma}, x, w) : \\ V(t_c, t_s, t_z, \vec{\sigma}, x, \pi) = 0 \text{ and } (x, w) \in R \text{ and } |\vec{\sigma} \setminus S| \geq t_c\Big] \approx 0,$$

where K on query i outputs $\sigma_i \leftarrow K(1^k)$ and sets $S := S \cup \{\sigma_i\}$.

Our protocols will have perfect (t_c, t_s, t_z, n)-completeness for all $0 \leq t_c \leq n$. In other words, even if the adversary chooses all common reference strings itself, we still have probability 1 of outputting an acceptable proof.

(t_c, t_s, t_z, n)-SOUNDNESS. The goal of the adversary in the soundness definition is to forge a proof using n common reference strings, even if t_s of them are honestly generated. The adversary gets to see possible choices of correctly generated common reference strings and can adaptively choose n of them, it may also in these n common reference strings include up to $n - t_s$ fake common reference strings that it chooses itself.

We say (K, P, V) is (t_c, t_s, t_z, n)-sound if for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\Pr\Big[S := \emptyset; (\vec{\sigma}, x, \pi) \leftarrow \mathcal{A}^K : V(t_c, t_s, t_z, \vec{\sigma}, x, \pi) = 1 \text{ and } x \notin L \text{ and } |\vec{\sigma} \setminus S| \geq t_s\Big] \approx 0,$$

where K is an oracle that on query i outputs $\sigma_i \leftarrow K(1^k)$ and sets $S = S \cup \{\sigma_i\}$.

(t_c, t_s, t_z, n)-ZERO-KNOWLEDGE. We wish to formulate that if t_z common reference strings are correctly generated, then the adversary learns nothing from the proof. As is standard in the zero-knowledge literature, we will say this is the case, when we can simulate the proof. Let therefore S_1 be an algorithm that outputs (σ, τ), respectively a simulation reference string and a simulation trapdoor. Let furtermore, S_2 be an algorithm that takes input $(t_c, t_s, t_z, \vec{\sigma}, \vec{\tau}, x, w)$ and simulates a proof π if $\vec{\tau}$ contains t_z simulation trapdoors for common reference strings in $\vec{\sigma}$.

We will strenghten the standard definition of zero-knowledge, by splitting the definition into two parts. The first part simply says that the adversary cannot distinguish real common reference strings from simulation reference strings. The second part, says that *even with access to the simulation trapdoors* the adversary cannot distinguish the prover from the simulator on a set of simulated reference strings. This kind of definition was considered by Groth in [Gro06] in the common reference string model and was proven to imply adaptive multi-theorem zero-knowledge.

We say (K, P, V) is (t_c, t_s, t_z, n)-composable zero-knowledge if there exists S_1, S_2 such that we have both reference string indistinguishability and simulation indistinguishability as described below. Either of these come in computational, statistical and perfect flavors. We describe the computational flavor, since that is the most relevant in this paper.

REFERENCE STRING INDISTINGUISHABILITY. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\Pr\left[\sigma \leftarrow K(1^k) : \mathcal{A}(\sigma) = 1\right] \approx \Pr\left[(\sigma, \tau) \leftarrow S_1(1^k) : \mathcal{A}(\sigma) = 1\right].$$

(t_c, t_s, t_z, n)-SIMULATION INDISTINGUISHABILITY. For all non-uniform interactive polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} & \Pr\left[S := \emptyset; (\vec{\sigma}, x, w) \leftarrow \mathcal{A}^{S_1}(1^k); \pi \leftarrow P(t_c, t_s, t_z, \vec{\sigma}, x, w) : \mathcal{A}(\pi) = 1 \text{ and } (x, w) \in R \text{ and } |\vec{\sigma} \setminus S| \geq t_z\right] \\ \approx\ & \Pr\left[S := \emptyset; (\vec{\sigma}, x, w) \leftarrow \mathcal{A}^{S_1}(1^k); \pi \leftarrow S_2(t_c, t_s, t_z, \vec{\sigma}, \vec{\tau}, x) : \mathcal{A}(\pi) = 1 \text{ and } (x, w) \in R \text{ and } |\vec{\sigma} \setminus S| \geq t_z\right], \end{aligned}$$

where $\vec{\tau}$ contains the simulation trapdoors corresponding to σ_i's generated by S_1.

LOWER BOUNDS FOR MULTI-STRING NIZK PROOFS. Soundness and zero-knowledge are complementary. The intuition is that if an adversary controls enough strings to simulate a proof, then he can prove anything and we can no longer have soundness.We capture this formally in the following theorem.

Theorem 1 *If L is a language with a proof system (K, P, V) that has (t_c, t_s, t_z, n)-completeness, soundness and zero-knowledge then $L \in \mathrm{P/poly}$ or $t_s + t_z > n$.*

Proof. Assume we have an (t_c, t_s, t_z, n)-NIZK proof system for R defining L and $t_s + t_n \leq n$. Given an element x, we wish to decide whether $x \in L$ or not. We simulate t_z common reference strings $(\sigma_i, \tau_i) \leftarrow S_1(1^k)$ and generate $n - t_z$ common reference strings $\sigma_j \leftarrow K(1^k)$ setting $\tau_j = \perp$. We then simulate the proof $\pi \leftarrow S_2(\vec{\sigma}, \vec{\tau}, x)$. Output $V(\vec{\sigma}, x, \pi)$.

Let us analyze this algorithm. If $x \in L$, then by (t_c, t_s, t_z, n)-completeness a prover with access to a witness w would output a proof that the verifier accepts if all common reference strings are generated correctly. By reference string indistinguishability, we will therefore also accept the proof when some of the common reference strings are simulated. By (t_c, t_s, t_z, n)-simulation indistinguishability, where we give (x, w) as non-uniform advice to $\mathcal{A}$, we will output 1 with overwhelming probability on $x \in L$.

On the other hand, if $x \notin L$, then by the (t_c, t_s, t_z, n)-soundness we output 0 with overwhelming probability, since $n - t_z \geq t_s$ common reference strings have been generated correctly. This shows that $L \in \mathrm{BPP/poly}$. By [Adl78] we have $\mathrm{P/poly} = \mathrm{BPP/poly}$, which concludes the proof. □

In general, we wish to minimize t_s to make it more probable that the protocol is sound, and at the same time we wish to minimize t_z to make it more probable that the protocol is zero-knowledge. In many cases, choosing n odd, and setting $t_s = t_z = \frac{n+1}{2}$ will be a reasonable compromise. However, there are also cases where it might be relevant to have an eskewed setting. Consider the case, where Alice wants to e-mail a NIZK proof to Bob, but does not know Bob's preferences with respect to common reference strings. She may pick a set of common reference strings and make a multi-string proof. Bob did not participate in deciding which common reference strings to use, however, if they came from trustworthy authorities he may be willing to accept that probably one of the authorities is honest. On the other hand, Alice gets to choose the authorities, so she may be wiling to believe that all of them are honest. The appropriate choice in this situation, is a multi-string proof with $t_s = 1, t_z = n$.

(t_c, t_s, t_z, n)-KNOWLEDGE. Strenghtening the definition of soundness, we call (K, P, V) a (t_c, t_s, t_z, n) proof of knowledge for R if there exists a knowledge extractor $E = (E_1, E_2)$ with the properties described below.

For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\Pr\left[\sigma \leftarrow K(1^k) : \mathcal{A}(\sigma) = 1\right] \approx \Pr\left[(\sigma, \xi) \leftarrow E_1(1^k) : \mathcal{A}(\sigma) = 1\right].$$

For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\Pr\Big[S := \emptyset; (\vec{\sigma}, x, \pi) \leftarrow \mathcal{A}^{E_1}(1^k); w \leftarrow E_2(t_c, t_s, t_z, \vec{\sigma}, \vec{\xi}, x, \pi) : \\ V(t_c, t_s, t_z, \vec{\sigma}, x, \pi) = 1 \text{ and } |\vec{\sigma} \setminus S| \geq t_s \text{ and } (x, w) \notin R\Big] \approx 0,$$

where E_1 is an oracle that returns $(\sigma, \xi) \leftarrow E_1$ and sets $S := S \cup \{\sigma\}$, and $\vec{\xi}$ is the n element vector that contains at least t_s ξ's corresponding to the σ's in $\vec{\sigma}$ generated by E_1.

(t_c, t_s, t_z, n)-SIMULATION-SOUNDNESS. In security proofs, it is often useful to simulate a proof for a false statement. However, seeing a simulated proof for a false statement might enable an adversary to generate more proofs for false statements. We say an NIZK proof is (t_c, t_s, t_z, n)-simulation-sound if an adversary cannot prove any false statement even after seeing simulated proofs of arbitrary statements.

More precisely, a (t_c, t_s, t_z, n)-NIZK proof system (K, P, V, S_1, S_2) is (t_c, t_s, t_z, n)-simulation-sound if for all non-uniform polynomial time adversaries we have

$$\Pr\Big[S := \emptyset; Q := \emptyset; (\vec{\sigma}, x, \pi) \leftarrow \mathcal{A}^{S_1, S_2'(\cdot,\cdot)}(1^k) : \\ (x, \pi) \notin Q \text{ and } x \notin L \text{ and } V(\vec{\sigma}, x, \pi) = 1 \text{ and } |\vec{\sigma} \setminus S| \geq t_s\Big] \approx 0,$$

where S_1 returns $(\sigma, \tau) \leftarrow S_1(1^k)$ and sets $S := S \cup \{\sigma\}$, and $S_2'(\vec{\sigma}, x)$ returns $\pi \leftarrow S_2(t_c, t_s, t_z, \vec{\sigma}, \vec{\tau}, x)$ with $\vec{\tau}$ containing simulation trapdoors for the σ's generated by S_1 and sets $Q := Q \cup \{x, \pi\}$.

(t_c, t_s, t_z, n)-SIMULATION-EXTRACTABILITY. Since we are working in the multi-string model, we assume strings can be set up and used by anybody who comes along. Knowledge extraction and zero-knowledge may both be very desirable properties, however, we may also imagine security proofs where we at the same time need to extract witnesses from some proofs and simulate other proofs. This joint simulation/extraction is for instance often seen in security proofs in the UC framework.

Combining simulation soundness and knowledge extraction, we may therefore require that even after seeing many simulated proofs, whenever the adversary makes a new proof we are able to extract a witness. We call this property simulation-extractability. Simulation-extractability implies simulation-soundness, because if we can extract a witness from the adversary's proof, then obviously the statement must belong to the language in question.

Consider a (t_c, t_s, t_z, n)-NIZK proof of knowledge $(K, P, V, S_1, S_2, E_1, E_2)$. Let SE_1 be an algorithm that outputs (σ, τ, ξ) such that it is identical to S_1 when restricted to the first two parts (σ, τ). We say the NIZK proof is (t_c, t_s, t_z, n)-simulation-extractable if for all non-uniform polynomial time adversaries we have

$$\Pr\Big[S := \emptyset; Q := \emptyset; (\vec{\sigma}, x, \pi) \leftarrow \mathcal{A}^{SE_1', S_2(\cdot,\cdot)}(1^k); w \leftarrow E_2(t_c, t_s, t_z, \vec{\sigma}, \vec{\xi}, x, \pi) : \\ (x, \pi) \notin Q \text{ and } (x, w) \notin R \text{ and } V(t_c, t_s, t_z, \vec{\sigma}, x, \pi) = 1 \text{ and } |\vec{\sigma} \setminus S| \geq t_s\Big] \approx 0,$$

where SE_1' outputs (σ, ξ) from $(\sigma, \tau, \xi) \leftarrow SE_1(1^k)$ and sets $S = S \cup \{\sigma\}$, S_2 outputs $\pi \leftarrow S_2(t_c, t_s, t_z, \vec{\sigma}, \vec{\tau}, x)$, where $\vec{\tau}$ contains t_z τ's corresponding to σ's generated by SE_1 and sets $Q = Q \cup \{x, \pi\}$, and $\vec{\xi}$ is a vector containing at least t_s ξ's generated by SE_1 corresponding to σ's in $\vec{\sigma}$.

(t_c, t_s, t_z, n)-EXTRACTION ZERO-KNOWLEDGE. Combining simulation soundness and knowledge extraction, we may also require that even after seeing many extractions, it should still be hard to distinguish real proofs and simulated proofs from one another. This definition resembles the definition of chosen ciphertext attack secure public key encryption.

Consider a (t_c, t_s, t_z, n) NIZK proof of knowledge $(K, P, V, S_1, S_2, E_1, E_2)$. Let SE_1 be an algorithm that outputs (σ, τ, ξ) such that it is identical to S_1 when restricted to the first two parts (σ, τ). We say the NIZK proof is (t_c, t_s, t_z, n)-extraction zero-knowledge if for all non-uniform interactive polynomial time adversaries we have

$$\Pr\Big[S := \emptyset; (\vec{\sigma}, x, w) \leftarrow \mathcal{A}^{SE_1', E_2(\cdot,\cdot)}(1^k); \pi \leftarrow P(t_c, t_s, t_z, \vec{\sigma}, x, w) :$$
$$\mathcal{A}^{E_2(\cdot,\cdot)}(\pi) = 1 \text{ and } (x, w) \in R \text{ and } |\vec{\sigma} \setminus S| \geq t_z\Big] \approx$$
$$\Pr\Big[S := \emptyset; (\vec{\sigma}, x, w) \leftarrow \mathcal{A}^{SE_1', E_2(\cdot,\cdot)}(1^k); \pi \leftarrow S_2(t_c, t_s, t_z, \vec{\sigma}, \vec{\tau}, x) :$$
$$\mathcal{A}^{E_2(\cdot,\cdot)}(\pi) = 1 \text{ and } (x, w) \in R \text{ and } |\vec{\sigma} \setminus S| \geq t_z\Big],$$

where SE_1' outputs (σ, τ) from $(\sigma, \tau, \xi) \leftarrow SE_1(1^k)$ and sets $S = S \cup \{\sigma\}$, E_2 outputs $w \leftarrow E_2(t_c, t_s, t_z, \vec{\sigma}, \vec{\xi}, x)$, when the query contains t_s σ's generated by SE_1 and π is not the challenge proof.

3 Multi-string NIZK Proofs based on General Assumptions

MULTI-STRING NIZK PROOFS. We start out with a simple construction of a multi-string NIZK proof that works for $t_c = 0$ and all choices of t_s, t_z, n so $t_s + t_z > n$. We use two tools in this construction, a pseudorandom generator and a zap. Recall, a zap is a two-round public coin witness-indistinguishable proof, where the verifier's first message is chosen at random and can be fixed once and for all and be reused in subsequent zaps.

A common reference string will consist of a random value r and an initial message σ for the zap. Given a statement $x \in L$, the prover makes zaps for

$$x \in L \quad \text{or} \quad \text{there are } t_z \text{ common reference strings where } r \text{ is a pseudorandom value.}$$

In the simulation, we create simulation reference strings as $r = \mathrm{prg}(\tau)$ enabling the simulator to make zaps without knowing a witness w for $x \in L$.

Common reference string: Generate $r \leftarrow \{0,1\}^{2k}; \sigma \leftarrow \{0,1\}^{\ell_{\mathrm{zap}}(k)}$. Output $\Sigma := (r, \sigma)$.

Proof: Given input $t_z, (\Sigma_1, \ldots, \Sigma_n)$, a statement x and a witness w so $(x, w) \in R$, we wish to prove $x \in L$. Using NP-reductions, we create a polynomial size circuit C that is satisfiable if and only if

$$x \in L \quad \text{or} \quad |\{r_i | \exists \tau_i \, : \, r_i = \mathrm{PRG}(\tau_i)| \geq t_z.$$

Chosen appropriately, NP-reductions are witness preserving, so we also reduce w to a witness W for C being satisfiable. For all n common reference strings, generate $\pi_i \leftarrow P_{\mathrm{zap}}(\sigma_i, C, W)$. Return the proof $\Pi := (\pi_1, \ldots, \pi_n)$.

Verification: Given n common reference strings $(\Sigma_1, \ldots, \Sigma_n)$, a statement x and a proof $\Pi = (\pi_1, \ldots, \pi_n)$ return 1 if and only if all of them satisfy $V_{\mathrm{zap}}(\sigma_i, C, \pi_i) = 1$, where C is generated as in the proof.

Simulated reference string: Select $\tau \leftarrow \{0,1\}^k; r := \mathrm{PRG}(\tau)$ and $\sigma \leftarrow \{0,1\}^{\ell_{\mathrm{zap}}(k)}$. Output $((r, \sigma), \tau)$.

Simulated proof: Given input $(\Sigma_1, \ldots, \Sigma_n), (\tau_1, \ldots, \tau_n), x$ so we have for t_z reference strings $r_i = \mathrm{PRG}(\tau_i)$ we wish to simulate a proof Π. As in a proof, use NP-reductions to get a circuit C that is satisfiable if and only if $x \in L$ or $|\{r_i | \exists \tau_i : r_i = \mathrm{PRG}(\tau_i)| \geq t_z$. Pick the first t_z common reference string Σ_i, where $r_i = \mathrm{PRG}(\tau_i)$, and reduce this to a witness W for the satisfiability of C. For all n common reference strings, generate $\pi_i \leftarrow P_{\mathrm{zap}}(\sigma_i, C, W)$. Return the simulated proof $\Pi := (\pi_1, \ldots, \pi_n)$.

Theorem 2 *The existence of one-way functions and zaps with perfect completeness imply the existence of $(0, t_s, t_z, n)$ NIZK proofs for any $1 \leq t_s, t_z \leq n$ with $t_s + t_z > n$ in the common random strings model with statistical $(0, t_s, t_z, n)$-soundness. In particular, enhanced trapdoor permutations imply the existence of NIZK proofs in the common random string model, which in turn implies the existence of zaps.*

Proof. Trapdoor permutations imply one-way functions, which in turn imply the existence of pseudorandom generators [HILL99]. Dwork and Naor [DN02] construct zaps from NIZK proofs in the random string model, which can be built from trapdoor permutations. There are a few details that are easy to resolve, but worth mentioning. First, they allow completeness error in the zaps, however, it is easy to see that their construction is actually perfectly complete if one uses an NIZK proof with perfect completeness in their construction. Second, their construction uses an inital message that is polynomial in the statement size, whereas we want the authorities to generate common reference strings without knowing the statement size in advance. Plugging in any NIZK with common random string size that is independent of the statement size circumvents this problem.

Direct verification reveals that we have perfect completeness, even for $t_c = 0$. Let us prove that we have $(0, t_s, t_z, n)$-soundness. Any honestly generated common reference string has negligible probability of containing a pseudorandom value r. With t_s honestly generated strings and $t_z > n - t_s$, there is negligible probability that $(\Sigma_1, \ldots, \Sigma_n)$ have t_z or more pseudorandom values. If $x \notin L$, the resulting circuit C is unsatisfiable. Also, at least one of the common reference strings has a correctly generated initial message for the zap. By the statistical soundness of this zap it is thus hard to construct a valid proof, even for an unbounded adversary of C being satisfiable.

We now turn to the question of $(0, t_s, t_z, n)$-zero-knowledge. Computational reference string indistinguishability follows from the pseudorandomness of PRG. With at least t_z simulated reference strings the only difference between proofs using the witness of $x \in L$ and simulated proofs using the simulation trapdoors is the witnesses we are using in the zaps. Computational simulation indistinguishability follows from a standard hybrid argument using the witness indistinguishability of the zaps. □

$(0, t_s, t_z, n)$-SIMULATION-EXTRACTABLE NIZK PROOF. More advanced proofs, such as multi-string NIZK proofs of knowledge that are simultaneously $(0, t_s, t_z, n)$-simulation-extractable and $(0, t_s, t_z, n)$-extraction zero-knowledge can also be constructed in the multi-string model.

To permit the extraction of witnesses, we include a public key for a CCA2-secure cryptosystem in the common reference strings. In a proof, the prover will make a (t_s, n)-threshold secret sharing of the witness and encrypt the shares under the n public keys. To extract the witness, we will decrypt t_s of these ciphertexts and combine the shares to get the witness.

To avoid tampering with the proof, we will use a strong one-time signature scheme. The prover generates a key $(vk_{\mathrm{sots}}, sk_{\mathrm{sots}}) \leftarrow K_{\mathrm{sots}}(1^k)$ that he will use to sign the proof. The implication is that the adversary, who sees simulated proofs, must still use a different vk_{sots} in his forged proof, because he cannot forge the strong one-time signature.

The common reference string will contain a value, which in a simulation string will be pseudorandom. The prover will prove that he encrypted a (t_s, n)-secret sharing of the witness, or that he knows how to evaluate t_z pseudorandom functions in vk_{sots} using the seeds of the respective common reference strings. On a real common reference string, this seed is not known and therefore he cannot make such a proof. On the other hand, in the simulation the simulator does know these seeds and can therefore simulate without knowing the witness. Simulation soundness follows from the adversary's inability to guess these pseudorandom functions on vk_{sots}, even if it knew the evaluations on many other verification keys.

Zero-knowledge under extraction attack follows from the CCA2-security of the cryptosystem. Even after having seen many extractions, the ciphertexts reveal nothing about the witness, or even whether the trapdoor has been used to simulate a proof.

Common reference string/simulation string: Generate $(pk_1, dk_1), (pk_2, dk_2) \leftarrow K_{\text{cca2}}(1^k); r \leftarrow \{0,1\}^{2k}; \sigma \leftarrow \{0,1\}^{\ell_{\text{zap}}(k)}$. Return $\Sigma := (pk_1, pk_2, r, \sigma)$.

The simulators and extractors S_1, E_1, SE_1 will generate the simulated reference strings in the same way, except for choosing $\tau \leftarrow \{0,1\}^k$ and $r := \text{PRF}_\tau(0)$. We use the simulation trapdoor τ and the extraction key $\xi := dk_1$.

Proof: $P(0, t_s, t_z, (\Sigma_1, \ldots, \Sigma_n), x, w)$ where $(x, w) \in R$ runs as follows. First, generate a key pair for a strong one-time signature scheme $(vk_{\text{sots}}, sk_{\text{sots}}) \leftarrow K_{\text{sots}}(1^k)$. Use (t_s, n)-threshold secret sharing to get shares $w_1, \ldots, w_n$ of w. Encrypt the shares as $c_{i1} = E_{pk_{i1}}(w_i, vk_{\text{sots}}; r_{i1})$. Also encrypt dummy values $c_{i2} \leftarrow E_{pk_{i2}}(0)$. Consider the statement: $c_{11}, \ldots, c_{n1}$ all encrypt vk_{sots}, and furthermore $c_{11}, \ldots, c_{n1}$ are encryptions of shares of a (t_s, n)-secret sharing of a witness w so $(x, w) \in R$ *or* at least t_z of the r_i's on the common reference strings are on the form $r_i = \text{PRF}_{\tau_i}(0)$ and the corresponding c_{i2} is an encryption of $\text{PRF}_{\tau_i}(vk_{\text{sots}})$. We can reduce this statement to a polynomial size circuit C and a satisfiability witness W. For all i's we create a zap $\pi_i \leftarrow P_{\text{zap}}(\sigma_i, C, W)$. Finally, we sign everything using the strong one-time signature $sig \leftarrow \text{Sign}_{sk_{\text{sots}}}(vk_{\text{sots}}, \Sigma_1, c_{11}, c_{12}, \pi_1, \ldots, \Sigma_n, c_{1n}, c_{2n}, \pi_n)$. The proof is $\Pi = (vk_{\text{sots}}, c_{11}, c_{12}, , \pi_1, \ldots, c_{1n}, c_{2n}, \pi_n, sig)$.

Verification: To verify Π on the form described above, verify the strong one-time signature and verify the n zaps $\pi_1, \ldots, \pi_n$.

Extraction: To extract a witness check that the proof is valid. Next, use the first t_s extraction keys in $\vec{\xi}$ to decrypt the corresponding t_s ciphertexts. Use Lagrange interpolation on the plaintexts to recover the witness w.

Simulated proof: To simulate a proof, pick the first t_z simulation trapdoors in $\vec{\tau}$. These are τ_i so $r_i = \text{PRF}_{\tau_i}(0)$. As in the proof generate $(vk_{\text{sots}}, sk_{\text{sots}}) \leftarrow K_{\text{sots}}(1^k)$. Create t_z pseudorandom values $v_i := \text{PRF}_{\tau_i}(vk_{\text{sots}})$. Encrypt the values as $c_{i2} \leftarrow E_{pk_{i2}}(v_i)$. For the other reference strings, just let $c_{i2} \leftarrow E_{pk_{i2}}(0)$. Let $w_1, \ldots, w_n$ be a (t_s, n)-threshold secret sharing of 0. We encrypt also these values as $c_{i1} \leftarrow E_{pk_{i1}}(w_i, vk_{\text{sots}})$. Let C be the circuit corresponding to the statement that $c_{11}, \ldots, c_{n1}$ contain vk_{sots}, and also $c_{11}, \ldots, c_{n1}$ contains a (t_s, n)-threshold secret sharing of a witness w so $(x, w) \in R$ or there are at least t_z of the ciphertexts $c_{12}, \ldots, c_{n2}$ that contain pseudorandom function evaluations on vk_{sots}. From the creation of the ciphertexts c_{i1} and c_{i2} we have a witness W for C being satisfiable. Create zaps $\pi_i \leftarrow P_{\text{zap}}(\sigma_i, C, W)$ for C being satisfiable. Finally, make a strong one-time signature on everything $sig \leftarrow \text{Sign}_{sk_{\text{sots}}}(vk_{\text{sots}}, \Sigma_1, c_{11}, c_{12}, \pi_1, \ldots, \Sigma_n, c_{n1}, c_{n2}, \pi_n)$. The simulated proof is $\Pi := (vk_{\text{sots}}, c_{11}, c_{12}, \pi_1, \ldots, c_{n1}, c_{n2}, \pi_n, sig)$.

Theorem 3 *The above protocol is a $(0, t_s, t_z, n)$-NIZK proof for all choices of $t_s + t_z > n$. It can be securely implented if trapdoor permutations exist, and it can be implemented with random strings if dense cryptosystems and enhanced trapdoor permutations exist.*

Proof. Let us start with the latter part. Enhanced trapdoor permutations, imply the existence of zaps with perfect completeness and pseudorandom functions and strong one-time signatures. Enhanced trapdoor permutations also imply the existence of CCA2-secure public key encryption with errorless decryption. In case dense public key cryptosystems and enhanced trapdoor permutations exist, CCA2-secure encryption with random strings as public keys exist.

Perfect completeness follows by direct verification. Common reference strings and simulated reference strings are indistinguishable by the pseudorandomness of the pseudorandom function PRF.

Let us consider extraction-sound zero-knowledge. The adversary knows the simulation trapdoors τ_i, and has access to an extraction oracle. He selects a statement x and a witness w and has to distinguish a proof on a simulated reference string using respectively the witness or the simulator. We consider a series of hybrid experiments.

Hybrid 1: This is the experiment, where we run the adversary on a simulated reference string and make proofs using the real prover and witness w.

Hybrid 2: We modify hybrid 1 by encrypting t_z pseudorandom values in $c_{12}, \ldots, c_{n2}$. We know t_z seeds τ_i such that $r_i = \mathrm{PRF}_{\tau_i}(0)$. Instead of setting $c_{i2} \leftarrow E_{pk_2}(0)$, we encrypt $c_{i2} \leftarrow E_{pk_2}(\mathrm{PRF}_{\tau_i}(vk_{\mathrm{sots}})$.

By the semantic security of the cryptosystem, hybrid 1 and hybrid 2 are computationally indistinguishable.

Hybrid 3: We modify hybrid 2, by reducing the pseudorandom values and the randomness used in forming the ciphertexts $c_{12}, \ldots, c_{n2}$ to form a witness W for C being satisfiable. We use this witness in the zaps, instead of the witness w.

By the witness-indistinguishability of the zaps, hybrid experiments 2 and 3 are indistinguishable.

Hybrid 4: We modify hybrid 3 such that if the adversary ever recycles one of the ciphertext c_{i1} from the challenge proof in one of the encryption queries and this is a valid proof, then we abort.

There is negligible probability of aborting. To make a valid proof, the adversary has to sign the proof using a verification key vk'_{sots}. By the existential forgeability of the strong one-time signature scheme, this verification key has to differ from the verification key vk_{sots} used in the challenge. This means, c_{i1} contains the wrong verification key. However, in the zaps, of which at least one is made using a correctly generated initial message, the adversary proves that c_{i1} does contain vk'_{sots}. By the soundness of the zap, there is negligible probability of the adversary succeeding in this.

Hybrid 5: We modify hybrid 4 by making a (t_s, n)-threshold secret sharing $w_1, \ldots, w_n$ of 0 instead of secret sharing w. We encrypt these shares in $c_{i1} \leftarrow E_{pk_{i1}}(w_i, vk_{\mathrm{sots}})$. This hybrid is identical to the simulation process.

Hybrid 4 and hybrid 5 are indistinguishable. We have ruled out that the adversary ever makes an extraction query, recycling a c_{i1} from the challenge. Using a hybrid argument on the chosen ciphertext attack security of the cryptosystems, the adversary cannot distinguish encryptions of shares of a threshold secret sharing of w from shares of a threshold secret sharing of 0. The remaining $n - t_z < t_s$ shares do not reveal anything.

Next, let us consider simulation-sound extractability. Here the adversary sees extraction keys, but not the simulation trapdoors of the common reference strings generated by SE_1. It has access to a simulation oracle, and in the end it outputs a statement and a proof. By the unforgeability of the strong one-time signature scheme, it cannot reuse a strong verification key vk_{sots} used in a simulated proof. Let us look at an honestly generated simulated common reference string. Since it does not know the seed for the pseudorandom function, it cannot encrypt a pseudorandom function evaluation of vk_{sots}. The zaps, of which at least one uses a correctly generated initial message, then tells us that $c_{11}, \ldots, c_{n1}$ contain a (t_s, n)-threshold secret sharing of w. Decrypting t_s of these ciphertexts, permits us to reconstruct the witness w.

A similar proof, shows that we have *statistical* $(0, t_s, t_z, n)$-knowledge extraction. The point in this proof is that with overwhelming probability a random string does not contain a pseudorandom value r, so therefore $c_{11}, \ldots, c_{n1}$ must encrypt a (t_s, n)-threshold secret sharing of a witness for $x \in L$. $\square$

4 Multi-string NIZK Proofs from Groups with a Bilinear Map

SETUP. We use groups $\mathbb{G}, \mathbb{G}_T$ of order p, where p is a k-bit prime. We make use of a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. I.e., for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$ we have $e(u^a, v^b) = e(u, v)^{ab}$. We require that $e(g, g)$ is a generator of $\mathbb{G}_T$ if g is a generator of $\mathbb{G}$. We require that group operations, group membership, and the bilinear map be efficiently computable. Such groups have been widely used in cryptography in recent years.

Let $\mathcal{G}$ be an algorithm that takes a security parameter as input and outputs $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ such that p is prime, $\mathbb{G}, \mathbb{G}_T$ are descriptions of groups of order p, $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is an admissible bilinear map as described above and g is a random generator of $\mathbb{G}$.

We use the decisional linear assumption introduced by Boneh, Boyen and Shacham [BBS04].

Definition 4 (Decisional Linear Assumption (DLIN)) *We say the decisional linear assumption holds for the bilinear group generator $\mathcal{G}$ if for all non-uniform polynomial time adversaries $\mathcal{A}$ we have*

$$\begin{aligned} & \Pr\Big[(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^k); x, y, r, s \leftarrow \mathbb{Z}_p : \mathcal{A}(p, \mathbb{G}, \mathbb{G}_T, e, g, g^x, g^y, g^{xr}, g^{ys}, g^{r+s}) = 1\Big] \\ \approx \ & \Pr\Big[(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^k); x, y, r, s, d \leftarrow \mathbb{Z}_p : \mathcal{A}(p, \mathbb{G}, \mathbb{G}_T, e, g, g^x, g^y, g^{xr}, g^{ys}, g^d) = 1\Big]. \end{aligned}$$

Throughout the paper, we work over a bilinear groups $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^k)$ generated such that the DLIN assumption holds for $\mathcal{G}$. Honest parties always check group membership of $\mathbb{G}, \mathbb{G}_T$ when relevant and halt if an element does not belong to a group that it was supposed to according to the protocol.

We will make some further assumptions on the groups that we use. Given a description of a group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ it should be possible to verify that indeed it is a group. Moreover, we will require that there is a decoding algorithm that given a random string of $(n+1)k$ bits interprets it as n random group elements. The decoding algorithm should be reversible, such that given n group elements we can create a random $(n+1)k$-bit string that will decode to the n group elements.

When working in the random strings model, we will also require that the group can be sampled from a random string of k-bit length.[1]

Example. We will offer a class of candidates for DLIN groups as described above. Consider the elliptic curve $y^2 = x^3 + 1 \bmod q$, where $q = 2 \bmod 3$ is a prime. It is straightforward to check that a point (x, y) is on the curve. Furthermore, picking $y \in \mathbb{Z}_q$ at random and computing $x = (y^2 - 1)^{\frac{q+1}{3}} \bmod q$ gives us a random point on the curve. The curve has a total of $q + 1$ points, including the point at infinity. When generating such groups, we will pick p as a random k-bit prime. We then let q be the smallest prime so $p|q+1$ and define $\mathbb{G}$ to be the order p subgroup of the curve. The target group is $\mathbb{G}_T = \mathbb{F}^*_{q^2}$ and the bilinear map is the modified Weyl-pairing. Verification of $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ being a group with bilinear maps is straightforward, since it corresponds to checking that p, q are primes so $p|q + 1$ and $q = 2 \bmod 3$ and g is an order p element on the curve.

PSEUDORANDOM GENERATORS FROM THE DLIN ASSUMPTION. Consider a DLIN group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$. Choose $x, y \leftarrow \mathbb{Z}^*_p$ at random and set $f = g^x, h = g^y$. Given random elements $u, v \leftarrow \mathbb{G}$, we can compute

[1] It is easy to modify our scheme to work with any group that can be specified by an $\mathcal{O}(k)$-bit random string.

$w = u^{1/x}v^{1/y}$. The DLIN assumption says that (f, h, u, v, w) is indistinguishable from (f, h, u, v, r), where r is a random group elements from $\mathbb{G}$. In other words, we can create a pseudorandom function $(x, y, u, v) \mapsto (g^x, g^y, u, v, u^{1/x}v^{1/y})$ that strecthes our randomness with an extra group element. We will need to create random looking strings that have hidden structure, this construction gives us exactly that. However, we need to stretch our random group elements into more group elements.

Let us pick m pairs $(x_i, y_i) \leftarrow \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ and create corresponding $f_i = g^{x_i}, h_i = g^{y_i}$. We can now stretch $2n$ group elements $u_1, v_1, \ldots, u_n, v_n$ with mn extra group elements by computing $w_{ij} := u_j^{1/x_i}v_j^{1/y_i}$.

It turns out that if the n pairs of group elements (u_j, v_j) are chosen at random, then $(f_1, h_1, \ldots, f_m, h_m, u_1, v_1, \ldots, u_n, v_n, w_{11}, \ldots, w_{mn})$ looks like a random $2m + 2n + mn$-tuple of group elements. To see this, consider the following hybrid experiment E_{IJ}, where we pick w_{ij} at random for pairs (i, j) where $i < I \vee (i = I \wedge j < J)$ and compute the rest of the w_{ij}'s according to the method described above. We need to prove that the w_{ij}'s generated in respectively E_{11} and $E_{m,n+1}$ are indistinguishable.

Consider first experiments $E_{I,J}, E_{I,J+1}$ for $1 \leq I \leq m, 1 \leq J \leq n$. In case there is a non-uniform adversary $\mathcal{A}$ that can distinguish these two experiments, then we can break the DLIN assumption as follows. We have a challenge (f, h, u, v, w) and wish to know whether $w = u^{1/x}v^{1/y}$ or w is random. We let $f_I := f, h_I := h$ and generate all the other f_i, h_i's according to the protocol. We set $u_J := u, v_J := v$ and $w_{IJ} := w$. For $i < I$ we pick w_{ij} at random. Also, for $i = I, j < J$ we pick w_{ij} at random. For $i = I, j > J$ we pick r_j, s_j at random and set $(u_j, v_j, w_{Ij}) = (f^{r_j}, h^{s_j}, g^{r_j+s_j})$. For $j < J$ we select (u_j, v_j) at random. Finally, for $i > I$ we compute all w_{ij} according to the protocol. If (u, v, w) is a linear tuple, we have the distribution from experiment $E_{I,J}$, whereas if (u, v, w) is a random tuple we have the distribution from experiment $E_{I,J+1}$. An adversary distinguishing these two experiments, therefore permits us to distinguish linear tuples from random tuples. We conclude the proof by observing $E_{I+1,1} = E_{I,n+1}$.

Observe, it is straightforward to provide a witness for (u, v, w) being a linear tuple. The witness consists of $\pi = u^{y/x}$. (u, v, w) is a linear tuple if and only if $e(u, h) = e(f, \pi)$ and $e(g, \pi v) = e(w, h)$. In other words, we can provide n^2 proofs π_{ij} for w_{ij} being correct. Furthermore, all these proofs consist of group elements and can be verified by checking a set of pairing product equations. It follows from Groth [Gro06] that there exists a (simulation-sound) NIZK proof of size $\mathcal{O}(mn)$ group elements for the w_{ij}'s having been computed correctly.

MULTI-STRING NIZK PROOFS FROM DLIN GROUPS. We will construct a protocol that is a $(0, t_s, t_z, n)$-simulation-sound NIZK proof for circuit satisfiability consisting of $\mathcal{O}((n + |C|)k)$ bits, where $|C|$ is the number of gates in the circuit and k is the security parameter. Typically, n will be much smaller than $|C|$, so the complexity matches the best known NIZK proofs for circuit satisfiability in the single common reference string model [GOS06b, GOS06a] that have proofs of size $\mathcal{O}(|C|k)$.

One could hope that the construction from the previous section could be implemented efficiently using groups with a bilinear map. This strategy does not work because each common reference string is generated at random and independently of the others. This means that even if the common reference strings contain descriptions of groups with bilinear maps, most likely they are different and incompatible groups.

In our construction, we accept that all the common reference strings describe different groups and we also let the prover pick a group with a bilinear map. Our solution to the problem described above, is to translate simulation reference strings into simulation reference strings in the prover's group. Consider a common reference string with group $\mathbb{G}_k$ and the prover's group $\mathbb{G}$. We will let the common reference string contain a random string r_k. From the earlier discussion, we know that we can build pseudorandom generators in each group. Consider now the pair of strings $(r_k \oplus s_k, s_k)$. Since strings can be interpreted as group elements, we have corresponding sets of group elements in respectively $\mathbb{G}_k$ and $\mathbb{G}$. However, since r_k is chosen at random it is unlikely that both $r_k \oplus s_k$ corresponds to a pseudorandom value in $\mathbb{G}_k$ and at the same time s_k corresponds to a pseudorandom value in $\mathbb{G}$. Of course, the prover has some degree of freedom in choosing the group $\mathbb{G}$, but if one is careful and chooses sufficient stretching length in the pseudorandom function one can use an entropy argument for it being unlikely that both strings are pseudorandom values.

Now we use non-interactive zaps and NIZK proofs to hop accross the bridge between the two groups. The prover will select s_k so $r_k \oplus s_k$ is a pseudorandom value in $\mathbb{G}_k$ specified by the common reference string and give an NIZK proof for this using that common reference string. In his own group, he gets n values $s_1, \ldots, s_k$ and proves that t_z of those are pseudorandom or C is satisfiable. In the simulation, on the other hand he knows the simulation trapdoors for t_z reference strings and he can therefore simulate NIZK proofs of $r_k \oplus s_k$ being pseudorandom. This means, he can select the corresponding s_k's as a pseudorandom values and use this to prove that there are at least t_z pseudorandom values in his own group, so he does not need to know the satisfiability witness w to carry out the proof in his own group.

There are more technical details to consider. We want to contruction to be efficient in n. Therefore, instead of proving directly that there are t_z pseudorandom values or C is satisfiable, we use a homomorphicly encrypted counter. In the simulation, we set the counter to be 1 for each pseudorandom value and to be 0 for the rest of the values in the prover's group. The homomorphic property enables us to multiply these ciphertexts and get an encrypted count of t_z. It is straightforward to prove that the count is t_z or C is satisfiable. As a further twist, we can set up the common reference strings such that they enable us to make simulation-sound NIZK proofs. This way, with a few extra tweaks we actually get a $(0, t_s, t_z, n)$-simulation-sound NIZK proof for circuit satisfiability when $t_s + t_z > n$.

Common reference string/simulation reference string: Generate a DLIN group $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^k)$. Generate a common reference string for a simulation-sound NIZK proof on basis of this group $\Sigma \leftarrow K_{\text{sim-sound}}(p, \mathbb{G}, \mathbb{G}_T, e, g)$ as in [Gro06]. Also, pick a random string $r \leftarrow \{0,1\}^{\ell_{65}(k)}$. Output $\Sigma := (p, \mathbb{G}, \mathbb{G}_T, e, g, \sigma, r)$.

Provided one can sample groups and group elements from random strings, this can all be set up in the random string model.

When generating a simulation reference string, use the simulator for the simulation-sound NIZK proof to generate $(\sigma, \tau) \leftarrow S_{\text{sim-sound}}(p, \mathbb{G}, \mathbb{G}_T, e, g)$. Output Σ as described above and simulation trapdoor τ.

Proof: Given common reference strings $(\Sigma_1, \ldots, \Sigma_n)$, a circuit C and a satisfiability witness w do the following. Pick a group $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^k)$. Pick also keys for a strong one-time signature scheme $(vk_{\text{sots}}, sk_{\text{sots}}) \leftarrow K_{\text{sots}}(1^k)$. Encode vk_{sots} as a tuple of $\mathcal{O}(1)$ group elements from $\mathbb{G}$.[2]

For each common reference string Σ_k do the following. Pick a pseudorandom value with 6 key pairs, 6 input pairs and 36 structured elements. This gives us 60 group elements from $\mathbb{G}_k$. Concatenate the tuple of 60 group elements with vk_{sots} to get $\mathcal{O}(1)$ group elements from $\mathbb{G}_k$. Make a simulation-sound NIZK proof, using σ_k, for these $\mathcal{O}(1)$ group elements being of a form such that the first 60 of them constitute a pseudorandom value. From [Gro06] we know that the size of this proof is $\mathcal{O}(1)$ group elements from $\mathbb{G}_k$. Define $s_k \in \{0,1\}^{65k}$ to be a random string such that $r_k \oplus s_k$ parses to the 60 elements from the pseudorandom value.

From now on we will work in the group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ chosen by the prover. Pick $pk := (f, h)$ as two random group elements. This gives us a CPA-secure cryptosystem, encrypting a message $m \in \mathbb{G}$ with randomness $r, s \in \mathbb{Z}_p$ as $E_{pk}(m; r, s) := (f^r, h^s, g^{r+s}m)$. For each $k = 1, \ldots, n$ we encrypt $1 = g^0$ as $c_k \leftarrow E_{pk}(1)$. Also, we take s_k and parse it as 60 group elements. Call this tuple z_k.

Make a non-interactive zap π using the group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ and combining techniques of [GOS06a]

[2]Observe, in DLIN groups the discrete logarithm problem is hard and therefore we can construct collision-free hash-functions, so there is no loss of generality in assuming the strong one-time signature scheme consists of a constant number of group elements.

and [Gro06] for the following statement:

$$C \text{ satisfiable} \quad \vee \quad (\prod_{k=1}^{n} c_k \text{ encrypts } g^t$$
$$\wedge\ \forall k : c_k \text{ encrypts } g^0 \text{ or } g^1\ \wedge\ \big(z_k \text{ is a pseudorandom structure } \vee c_i \text{ encrypts } g^0\big)\big).$$

The zap consists of $\mathcal{O}(n + |C|)$ group elements and has perfect soundness.

Sign everything $sig \leftarrow \text{Sign}_{sk_{\text{sots}}}(vk_{\text{sots}}, C, \Sigma_1, s_1, \pi_1, c_1, \ldots, \Sigma_n, s_n, \pi_n, c_n, p, \mathbb{G}, \mathbb{G}_T, e, g, f, h, \pi)$.

The proof is $\Pi := (vk_{\text{sots}}, s_1, \pi_1, c_1, \ldots, s_n, \pi_n, c_n, p, \mathbb{G}, \mathbb{G}_T, e, g, f, h, \pi, sig)$.

Verification: Given common reference strings $\Sigma_1, \ldots, \Sigma_n$, a circuit C and a proof as described above, do the following. For all k check the simulation-sound NIZK proofs π_k for $r_k \oplus s_k$ encoding a pseudorandom structure in $\mathbb{G}_k$ using common reference string σ_k. Verify $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ is a group with a bilinear map. Verify the zap π. Verify the strong one-time signature on everything. Output 1 if all checks are ok.

Simulated proof: We are given reference strings $\Sigma_1, \ldots, \Sigma_n$. t_z of them are simulation strings, where we know the simulation trapdoors τ_k for the simulation-sound NIZK proofs. We wish to simulate a proof for a circuit C being satisfiable.

We start by choosing a group $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}(1^k)$ and public key $f, h \leftarrow \mathbb{G}$. We create ciphertexts $c_k \leftarrow E_{pk}(g^1)$ for the t_z simulation reference strings, where we know the trapdoor τ_k, and set $c_k \leftarrow E_{pk}(g^0)$ for the rest. We also choose a strong one-time signature key pair $(vk_{\text{sots}}, sk_{\text{sots}}) \leftarrow K_{\text{sots}}^{(} 1^k)$.

For t_z of the common reference strings, we know the simulation key τ_k. This permits us to choose an arbitrary string s_k and simulate a proof π_k that $r_k \oplus s_k$ encodes a 60 element pseudorandom structure. This means, we are free to choose s_k so it encodes a pseudorandom struture z_k in $\mathbb{G}^{60}$. For the remaining $n - t_z < t_s$ reference strings, we select s_k so $r_k \oplus s_k$ does encode a pseudorandom struture in $\mathbb{G}_k$ and carry out a real simulation-sound NIZK proof π_k for it being a pseudorandom structure concatenated with vk_{sots}.

For all k we have c_k encrypting g^b, where $b \in \{0, 1\}$. We have $\prod_{k=1}^{n} c_k$ encrypting g^t. We also have for the t_z simulation strings, where we know τ_k that s_k encodes a pseudorandom structure, whereas for the other common reference strings we have c_k encrypts g^0. This means we can create the non-interactive zap π without knowing C's satisfiability witness.

Sign everything $sig \leftarrow \text{Sign}_{sk_{\text{sots}}}(vk_{\text{sots}}, C, \Sigma_1, s_1, \pi_1, c_1, \ldots, \Sigma_n, s_n, \pi_n, c_n, p, \mathbb{G}, \mathbb{G}_T, e, g, f, h, \pi)$.

The simulated proof is $\Pi := (vk_{\text{sots}}, s_1, \pi_1, c_1, \ldots, s_n, \pi_n, c_n, p, \mathbb{G}, \mathbb{G}_T, e, g, f, h, \pi, sig)$.

Theorem 5 *Assuming we have a DLIN group as described above, then the construction above gives us a $(0, t_s, t_z, n)$-simulation-sound NIZK proof for circuit satisfiability, where the proofs have size $\mathcal{O}((n+|C|)k)$ bits. The proof has statistical $(0, t_s, t_z, n)$-soundness. The scheme can be set up in the common random string model if we can sample groups with bilinear maps and group elements from random strings.*

Proof. We have already argued in the construction that if we can sample groups and group elements from random strings and vice versa given groups and group elements sample random strings that yield these group elements, then the common reference strings can be set up in the random strings model. Perfect completeness follows by straightforward verification.

Let us prove that we have statistical $(0, t_s, t_z, n)$-soundness. Consider first an arbitrary group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ chosen by the prover. By assumption, it can be verified that this describes a group with a bilinear map.

We will now bound the probability of both $r_k \oplus s_k$ and s_k specifying pseudorandom values in their respective groups for a random choice of r_k. Consider first the probability that a random string s_k specifies a pseudorandom value in $\mathbb{G}^{60}$. There are at most 2^{24k} pseudorandom strings, since the 12 pairs (f_i, h_i) and the 12 pairs (u_j, v_j) fully define the pseudorandom value. 60 random group elements have at $59k$ bits of entropy, so we get a probability of at most $2^{24k-59k} = 2^{-35k}$ of s_k specifying a pseudorandom value in $\mathbb{G}^{60}$. Similarly, for a random choice of r_k we have at most probability 2^{-35k} that $r_k \oplus s_k$ is a pseudorandom value in the group specified by the common reference string. With r_k, s_k both chosen at random, we have a total maximal probability of 2^{-70k} of both $r_k \oplus s_k$ and s_k specifying pseudorandom values. The prover can choose the group freely, giving him at most 2^{2k} different choices for the group $\mathbb{G}$ and g. He can also choose s_k freely, giving him 2^{65k} possibilities. Since r_k is chosen at random, there is at most probability $2^{2k+65k-70k} = 2^{-3k}$ of it being possible to choose s_k and the group $\mathbb{G}$ so both $r_k \oplus s_k$ and s_k specify pseudorandom values. With overwhelming probability, we can therefore assume that no honestly generated common reference string exists such that both $r_k \oplus s_k$ and s_k specify pseudorandom values in respectively $\mathbb{G}_k$ and $\mathbb{G}$.

Any common reference string Σ_k that is honestly generated has overwhelming probability of having a common reference string σ_k for the simulation-sound NIZK with perfect soundness. Whenever the prover makes a proof using this string, he must therefore pick s_k so $r_k \oplus s_k$ is pseudorandom. Consequently, s_k does not specify a pseudorandom value in the group $\mathbb{G}$. The zap has perfect soundness, so it shows that C is satisfiable or c_k contains g^0. Similarly, for any string Σ_k that is not honestly generated, the zap demonstrates that C is satisfiable or c_k contains g^0 or g^1. Since at least $t_s > n - t_z$ strings are honestly generated, we see that if C is unsatisfiable, then $\prod_{k=1}^{n} c_k$ contains one of the values $g^0, \ldots, g^{t_z-1}$. The zap therefore shows us that C must be satisfiable.

To argue computational $(0, t_s, t_z, n)$-simulation-soundness, observe that simulated proofs are signed with a strong one-time signature. Since the signature scheme has existential unforgeability, the adversary must choose a different vk_{sots} that it has not seen in a simulation. Recall, whenever we make a simulation-sound NIZK using a particular common reference string Σ_k, we concatenate vk_{sots} to $r_k \oplus s_k$ to get the statement we wish to prove. By the simulation-soundness of the NIZK proofs on honestly generated strings, we can not forge such a proof even though we have already seen simulated proofs. Therefore, $r_k \oplus s_k$ must be a pseudorandom string. We can now argue $(0, t_s, t_z, n)$-simulation-soundness just as we argued $(0, t_s, t_z, n)$-soundness.

It remains to prove computational $(0, t_s, t_z, n)$-zero-knowledge. Reference string indistinguishability follows from the reference string indistinguishability of the simulation-sound NIZK proofs. We will now consider simulation indistinguishability, so consider a case where the adversary sees simulated reference strings and gets the simulation trapdoors that allow the simulation of proofs for the reference strings. The adversary, chooses a set of common reference strings and receives a proof generated with the satisfiability witness for C or alternatively a simulated proof and wants to distinguish between the two possibilities.

Let us start with a simulated proof and compare it with a hybrid experiment, where we use the satisfiability witness for C in the non-interactive zap. By the computational witness-indistinguishability of the zap, the adversary cannot tell these two experiments apart. Next, let us choose all c_k's as encryptions of g^0. By the semantic security of the cryptosystem, the adversary cannot detect this change. We already select s_k so $r_k \oplus s_k$ specifies a pseudorandom value for the reference strings not generated by S_1. Let us switch to also selecting s_k so $r_k \oplus s_k$ specify a pseudorandom value in the common reference strings where we do know the simulation trapdoor. By the pseudorandomness of the strings, the adversary cannot detect this change either. Finally, instead of simulating the proofs for $r_k \oplus s_k$ specifying a pseudorandom value in $\mathbb{G}_k$, let us make a real proof. By the composable zero-knowledge property of the simulated reference strings for the simulation-sound NIZK proofs, the adversary cannot distinguish here either. With this last modification, we have actually ended up constructing proofs exactly as a real prover with access to a satisfiability witness does, so we have $(0, t_s, t_z, n)$ composable zero-knowledge. $\square$

5 Multi-party Computation

5.1 The UC Framework

The universal composability (UC) framework, see [Can01] for a detailed description, is a strong security model capturing security of a protocol under concurrent execution of arbitrary protocols. We model everything not directly related to the protocol through an environment $\mathcal{Z}$. The environment can at its own choosing give inputs to the parties running the protocol, and according to the protocol specification, the parties can give outputs to the environment. In addition, there is an adversary $\mathcal{A}$ that attacks the protocol. $\mathcal{A}$ can communicate freely with the environment. It can aadaptively corrupt parties, in which case it learns the entire history of that party and gains complete control over the actions of this party. The environment learns whenever a party is corrupted.

To model security we use a simulation paradigm. We specify the functionality $\mathcal{F}$ that the protocol should realize. The functionality $\mathcal{F}$ can be seen as a trusted party that handles the entire protocol execution and tells the parties what they would output if they executed the protocol correctly. In the ideal process, the parties simply pass on inputs from the environment to $\mathcal{F}$ and whenever receiving a message from $\mathcal{F}$ they output it to the environment. In the ideal process, we have an ideal process adversary $\mathcal{S}$. $\mathcal{S}$ does not learn the content of messages sent from $\mathcal{F}$ to the parties, but is in control of when, if ever, a message from $\mathcal{F}$ is delivered to the designated party. $\mathcal{S}$ can corrupt parties, at the time of corruption it will learn all inputs the party has received and all outputs it has sent to the environment. As the real world adversary, $\mathcal{S}$ can freely communicate with the environment.

We now compare these two models and say that the protocol securely realizes $\mathcal{F}$ if no environment can distinguish between the two worlds. This means, the protocol is secure, if for any polynomial time $\mathcal{A}$ running in the real world, there exists a polynomial time $\mathcal{S}$ running in the ideal process with $\mathcal{F}$, so no non-uniform polynomial time environment can distinguish between the two worlds.

Our goal in this section is to show that any well-formed functionality can be securely realized in the multi-string model. By well-formed functionality, we mean a functionality that is oblivious of corruptions of parties, runs in polynomial time, and in case all parties are corrupted it reveals the internal randomness used by the functionality is revealed to the ideal process adversary. This class contains all functionalities, we can reasonably expect to implement with multi-party computation, because an adversary can always corrupt a party and just have it follow the protocol, in which case the other parties in the protocol would never learn that it was corrupted.

5.2 Tools

This section will present a number of tools we will need in our constructions.

PSEUDORANDOM CRYPTOSYSTEM WITH PSEUDORANDOM KEYS. A cryptosystem $(K_{\text{pseudo}}, E, D)$ has pseudorandom ciphertexts of length $\ell_E(k)$ if for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} & \Pr\left[(pk, dk) \leftarrow K_{\text{pseudo}}(1^k) : \mathcal{A}^{E_{pk}(\cdot)}(pk) = 1\right] \\ \approx\ & \Pr\left[(pk, dk) \leftarrow K_{\text{pseudo}}(1^k) : \mathcal{A}^{R_{pk}(\cdot)}(pk) = 1\right], \end{aligned} \tag{1}$$

where $R_{pk}(m)$ runs $c \leftarrow \{0,1\}^{\ell_E(k)}$ and returns c. We require that the cryptosystem have errorless decryption.

Trapdoor permutations imply pseudorandom cryptosystems, since we can use the Goldreich-Levin hardcore bit [GL89] of a trapdoor permutation to make a one-time pad. For setting up our scheme in the common random string model, we will require that the cryptosystem has a pseudorandom public key as well. Pseudorandom cryptosystems with pseudorandom keys can be built from various assumption such as RSA, DDH and DLIN.

TAG-BASED SIMULATION-SOUND TRAPDOOR COMMITMENT. A tag-based commitment scheme has four algorithms. The key generation algorithm $K_{\mathrm{tag-com}}$ produces a commitment key ck as well as a trapdoor key tk. There is a commitment algorithm that takes as input the commitment key ck, a message m and any tag tag and outputs a commitment $c = \mathrm{Com}_{ck}(tag; m; r)$. To open a commitment c with tag tag we reveal m and the randomness r. Anybody can now verify $c = \mathrm{Com}_{ck}(tag; m; r)$. As usual, the commitment scheme must be both hiding and binding.

In addition, to these two algorithms there are also a couple of trapdoor algorithms $\mathrm{Tcom}, \mathrm{Topen}$ that allow us to create an equivocal commitment and later open this commitment to any value we prefer. We create an equivocal commitment and an equivocation key as $(c, ek) \leftarrow \mathrm{Tcom}_{tk}(tag)$. Later we can open it to any message m as $r \leftarrow \mathrm{Topen}_{ek}(tag; m)$, such that $c = \mathrm{Com}_{ck}(tag; m; r)$.

We require that equivocal commitments and openings are indistinguishable from real openings. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} & \Pr\left[(ck, tk) \leftarrow K_{\mathrm{tag-com}}(1^k) : \mathcal{A}^{\mathcal{R}(\cdot,\cdot)}(ck) = 1\right] \\ \approx\ & \Pr\left[(ck, tk) \leftarrow K_{\mathrm{tag-com}}(1^k) : \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(ck) = 1\right], \end{aligned} \tag{2}$$

where $\mathcal{R}(m, tag)$ returns a randomly selected randomizer and $\mathcal{O}(m, tag)$ computes $(c, ek) \leftarrow \mathrm{Tcom}_{tk}(tag, m); r \leftarrow \mathrm{Topen}_{ek}(tag, m)$ and returns r. Both oracles ignore tags that have already been submitted once.

The tag-based simulation-soundness property means that a commitment using tag remains binding even if we have made equivocations for commitments using different tags. For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} \Pr\Big[&(ck, tk) \leftarrow K_{\mathrm{tag-com}}(1^k); (tag, c, m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(ck) : tag \notin Q \text{ and} \\ & c = \mathrm{Com}_{ck}(tag; m_0; r_0) = \mathrm{Com}_{ck}(tag; m_1; r_1) \text{ and } m_0 \neq m_1\Big] \approx 0, \end{aligned} \tag{3}$$

where $\mathcal{O}(\mathrm{Com}, tag)$ computes $(c, ek) \leftarrow \mathrm{Tcom}_{tk}(tag)$, returns c and stores (c, tag, ek), and $\mathcal{O}(\mathrm{Open}, c, m, tag)$ returns $r \leftarrow \mathrm{Topen}_{ck}(tag, ek, c, m)$ if (c, tag, ek) has been stored, and where Q is the list of tags for which equivocal commitments have been made by $\mathcal{O}$.

The term tag-based simulation commitment comes from Garay, MacKenzie and Yang [GMY03], while the definition presented here is from MacKenzie and Yang [MY04]. The latter paper offers a construction based on one-way functions. In addition, since we are working over random strings, we want $K_{\mathrm{tag-com}}$ to output public keys that are random or pseudorandom, i.e., we can use $pk \leftarrow \{0,1\}^{\ell_{\mathrm{tag-com}}}$ to generate the public key.

EXTRACTABLE TRAPDOOR COMMITMENT SCHEME. We will need something that is stronger than tag-based simulation-sound commitments, namely a tag-based simulation-extractable commitment. This is a tag-based simulation-sound trapdoor commitment scheme with an additional algorithm $\mathrm{Extract}$ that given the trapdoor is able to extract the message inside the commitment. More precisely, with the trapdoor we can make trapdoor commitments, however, for all other tags, the adversary will end up making unconditionally binding commitments.

A tag-based simulation-extractable commitment scheme consists of five polynomial time algorithms $(K_{\mathrm{se-com}}, \mathrm{Com}, \mathrm{Tcom}, \mathrm{Topen}, \mathrm{Extract})$, such that the first 4 constitute a tag-based trapdoor commitment scheme, and such that $(K_{\mathrm{se-com}}, \mathrm{Com}, \mathrm{Extract})$ is a semantically secure cryptosystem. It will have the property that a non-uniform adversary with access to trapdoor openings of commitments and the extraction key, still cannot create a new commitment and opening thereof, such that the message it opens to differs from the extracted message.

For all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\Pr[Q := \emptyset; (\sigma,\tau,\xi) \leftarrow K_{\mathrm{se-com}}(1^k); (m,r) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}; c := \mathrm{Com}_\sigma(tag; m; r) : \\ \mathrm{Extract}_\xi(tag,c) \neq m \text{ and } tag \notin Q] \approx 0,$$

where $\mathcal{O}$ is an oracle that on input (tag, m) runs $(c, ek) \leftarrow \mathrm{Tcom}_\tau(tag); r \leftarrow \mathrm{Topen}_{ek}(tag, m)$, returns r and sets $Q := Q \cup \{tag\}$.

We will construct a tag-based simulation-extractable commitment scheme from the tools in this section. We use a tag-based simulation-sound trapdoor commitment scheme to commit to each bit of m. If m has length ℓ this gives us commitments $c_1, \ldots, c_\ell$. When making trapdoor commitments, we can use the trapdoor key tk to create equivocal commitments $c_1, \ldots, c_\ell$ that can be opened to any bit we like.

We still have an extraction problem, we may be unable to extract a message from tag-based commitments created by the adversary. To solve this problem we choose to encrypt the openings of the commitments. Now we can extract messages, but we have reintroduced the problem of equivocation. In a trapdoor commitment we may know two different openings of a commitment c_i to respectively 0 and 1, however, if we encrypt the opening then we are stuck with one possible opening. This is where the pseudorandomness property of the cryptosystem comes in handy. We can simply make two encryptions, one of an opening to 0 and one of an opening to 1. Since the ciphertexts are pseudorandom, we can open the ciphertext containing the opening we want and claim that the other ciphertext was chosen as a random string. To recap, the idea so far to commit to a bit b is to make a tag-based simulation-sound trapdoor commitment c_i to this bit, and create a ciphertext $c_{i,b}$ containing an opening of c_i to b, while choosing $c_{i,1-b}$ as a random string.

These are the main ideas, we now present the protocol in Figure 1.

Theorem 6 *Tag-based simulation-extractable commitment schemes exist with pseudorandom keys if pseudorandom cryptosystems with pseudorandom keys exist.*

Proof. Tag-based simulation-sound trapdoor commitments with pseudorandom keys can be built from one-way functions, so we have the tools needed in the construction. This also shows that we have pseudorandom keys for the tag-based simulation-extractable commitment scheme.

We now need to prove that even after seeing trapdoor commitments and openings, it is hard to come up with a commitment with a different tag, where the opening and extraction are different. Consider first the case, where the adversary for some index i creates c_i, c_{i0}, c_{i1} so both c_{i0} and c_{i1} decrypt to valid openings of c_i to respectively 0 and 1. Since tag has not been used before, we have not used tag, i in any commitment we have trapdoor opened before, so we have broken the simulation-sound binding property of the tag-based simulation-sound trapdoor commitment. The errorless decryption property of the pseudorandom cryptosystem now tells us that if the adversary opens all triples c_i, c_{i0}, c_{i1} succesfully, then so must we get these openings when decrypting.

We also need to prove that we have the trapdoor property. We will modify the trapdoor oracle in several steps and show that $\mathcal{A}$ cannot tell the difference. Let us start with the oracle that on input (tag, m) returns a randomly chosen randomizer $r_1, R_{1,m_1}, c_{1,1-m_1}, \ldots, r_\ell, R_{\ell,m_\ell}, c_{1,1-m_b}$. Instead of making commitments $c_i := \mathrm{Com}_{ck}(tag, i; m_i; r_i)$, we may instead run $(c_i, ek_i) \leftarrow \mathrm{Tcom}_{tk}(tag, i); r_i \leftarrow \mathrm{Topen}_{ek_i}(m_i)$ and use r_i as the randomizer. By the trapdoor property of the tag-based simulation-sound commitment the two oracles are indistinguishable to $\mathcal{A}$.

Next, consider the trapdoor oracle, where we make trapdoor openings to both r_{i0} and r_{i1} so $c_i = \mathrm{Com}_{ck}(tag, i; b; r_{i,b})$ for both $b = 0$ and $b = 1$. We encrypt $(b, r_{i,b})$ with randomness $R_{i,b}$. We then return $r_i, R_{i,m_i}, c_{i,1-m_i}$. By the pseudorandomness of the ciphertexts, this is indistinguishable from the previous oracle. □

STRONG ONE-TIME SIGNATURES. We remind the reader that strong one-time signatures allow a non-uniform polynomial time adversary to ask an oracle for a signature on one arbitrary message. Under this

Random key: Return $\sigma := (ck, pk) \leftarrow \{0,1\}^{\ell_{\text{tag}-\text{com}}(k)} \times \{0,1\}^{\ell_{\text{pseudo}}(k)}$

Simulation-extraction key:

1. $(ck, tk) \leftarrow K_{\text{tag}-\text{com}}(1^k)$
2. $(pk, xk) \leftarrow K_{\text{pseudo}}(1^k)$
3. Return $\sigma = (ck, pk)$, $\tau = (\sigma, tk)$, $\xi = (\sigma, xk)$

Commitment: On input (σ, tag, m) and randomizers as described below do

1. For $i = 1$ to ℓ select r_i at random and let $c_i := \text{Com}_{ck}(tag, i; m_i; r_i)$
2. For $i = 1$ to ℓ select R_{i,m_i} at random and set $c_{i,m_i} = E_{pk}(m_i, r_i; R_{i,m_i})$ and choose $c_{i,1-m_i}$ as a random string.
3. Return $c := (c_1, c_{10}, c_{11}, \ldots, c_\ell, c_{\ell 0}, c_{\ell 1})$

Opening: On input $(tag, c, m, r_1, R_{1,m_1}, \ldots, r_\ell, R_{\ell,m_\ell})$ do

1. Verify that for all i we have $c_i = \text{Com}_{ck}(tag, i; m_i; r_i)$
2. Verify that for all i we have $c_{i,m_i} = E_{pk}(m_i, r_i; R_{i,m_i})$
3. Return 1 if all checks work out, else return 0

Trapdoor commitment: On input $\tau = (tk, pk)$ do

1. For $i = 1$ to ℓ let $(c_i, e_i) \leftarrow \text{Tcom}_{tk}(tag, i)$ and let $r_{i,0}, r_{i,1}$ be equivocations so $c_i = \text{Com}_{ck}(tag, i; b; r_{i,b})$ for $b \in \{0,1\}$.
2. For $i = 1$ to ℓ select randomness $R_{i,b}$ and set $c_{i,b} := E_{pk}(b, r_{i,b}; R_{i,b})$.
3. Return $c := (c_1, c_{10}, c_{11}, \ldots, c_\ell, c_{\ell 0}, c_{\ell 1})$ and $ek := (\sigma, r_1, R_{1,0}, R_{1,1}, \ldots, r_\ell, R_{\ell,0}, R_{\ell,1})$.

Trapdoor opening: On input (tag, ek, c, m) return $(r_1, R_{1,m_1}, \ldots, r_\ell, R_{\ell,m_\ell})$.

Extraction: On input $(tag, (\sigma, dk), c)$ use the decryption key to decrypt the ciphertexts c_{ib}. In case, we for i have exactly one ciphertext c_{ib} that decrypts to b, r_{ib} so $c_i = \text{Com}_\sigma(tag, i; b; r_{ib})$, we set $m_i := b$. In case all these processes succeed, we return the concatenation m, else we return $\perp$.

Figure 1: Tag based simulation-extractable commitment.

attack, it must be infeasible to forge a signature on any different message and infeasible to come up with a different signature on the same message. Strong one-time signatures can be constructed from one-way functions.

5.3 Multi-party Computation in the Multi-string Model

We will demonstrate that any well-formed functionality $\mathcal{F}$ can be securely realized in the multi-string model. In this proof, we build on a result by Canetti et al. [CLOS02], which demonstrates that for any well-formed functionality $\mathcal{F}$ there is a non-trivial protocol that securely realizes it in the common random string model. Our task can therefore be simplified to securely realizing $\mathcal{F}_{\text{CRS}}$ in the multi-string model.

Let us first formalize the multi-string model in the UC framework. Figure 2 gives an ideal multi-string functionality $\mathcal{F}_{\text{MCRS}}$. We will construct universally composable commitments, see Figure 3, in the multi-string model. Next, we will show that the ideal common random string generator $\mathcal{F}_{\text{CRS}}$, see Figure 4, can be securely realized in the $(\mathcal{F}_{\mathrm{COM}}^{1:N})$-hybrid model.

Functionality $\mathcal{F}_{\mathrm{MCRS}}$

Parametrized by polynomial ℓ_{mcrs}, and running with parties $P_1, \ldots, P_N$ and adversary $\mathcal{S}$.

String generation: On input $(\mathbf{crs}, sid)$ from $\mathcal{S}$, pick $\sigma \leftarrow \{0,1\}^{\ell_{\mathrm{mcrs}}(k)}$ and store it. Send $(\mathbf{crs}, sid, \sigma)$ to $\mathcal{A}$.

String selection: On input $(\mathbf{vector}, sid, \sigma_1, \ldots, \sigma_n)$ where $\sigma_1, \ldots, \sigma_n \in \{0,1\}^{\ell_{\mathrm{mcrs}}(k)}$ from $\mathcal{S}$ check that more than half of the strings $\sigma_1, \ldots, \sigma_n$ match stored strings. In that case output $(\mathbf{vector}, sid, \sigma_1, \ldots, \sigma_n)$ to all parties and halt.

Figure 2: The ideal multi-string generator.

Functionality $\mathcal{F}_{\mathrm{COM}}^{1:N}$

Parametrized by polynomial ℓ, and running with parties $P_1, \ldots, P_N$ and adversary $\mathcal{S}$.

Commitment: On input $(\mathbf{commit}, sid, m)$ from party P_i check that $m \in \{0,1\}^{\ell(k)}$ and in that case store (sid, P_i, m) and send $(\mathbf{commit}, sid, P_i)$ to all parties and $\mathcal{S}$. Ignore future $(\mathbf{commit}, sid, \cdot)$ inputs from P_i.

Opening: On input $(\mathbf{open}, sid)$ from P_i check that (sid, P_i, m) has been stored, and in that case send $(\mathbf{open}, sid, P_i, m)$ to all parties and $\mathcal{S}$.

Figure 3: The ideal commitment functionality.

Functionality $\mathcal{F}_{\mathrm{CRS}}$

Parameterized with polynomial ℓ and running with parties $P_1, \ldots, P_n$ and adversary $\mathcal{S}$.

CRS generation: Generate random $\sigma \leftarrow \{0,1\}^{\ell(k)}$ and output $(\mathbf{crs}, sid, \sigma)$ to all parties and $\mathcal{S}$. Halt.

Figure 4: The ideal common random string generator.

We will assume the parties can broadcast messages, i.e., have access to an ideal broadcast functionallity $\mathcal{F}_{\mathrm{BC}}$, see Figure 5. We note that broadcast can be securely realized in a constant number of rounds if authenticated communication is available [GL05]. Furthermore, authenticated communication can be securely realized using digital signatures, so one possible setup is that the parties somehow have managed to exchange verification keys for the digital signature scheme.

Functionality $\mathcal{F}_{\mathrm{BC}}$

Running with parties $P_1, \ldots, P_n$ and adversary $\mathcal{S}$.

Broadcast: On input $(\mathbf{broadcast}, sid, ssid, m)$ from P_i, send $(\mathbf{broadcast}, sid, ssid, P_i, m)$ to all parties and $\mathcal{S}$. Ignore future $(\mathbf{broadcast}, sid, ssid, \cdot)$ inputs from P_i.

Figure 5: The ideal authenticated broadcast fucntionality.

5.4 Universally Composable Commitment in the Multi-string Model

In our security proof, the ideal process adversary $\mathcal{S}$ will interact with $\mathcal{F}_{\mathrm{COM}}^{1:N}$ and make a black-box simulation of $\mathcal{A}$ running with $\mathcal{F}_{\mathrm{MCRS}}$ and $P_1, \ldots, P_N$. There are two general types of issues that can come up in the ideal process simulation. First, when $\mathcal{F}_{\mathrm{COM}}^{1:N}$ tells $\mathcal{S}$ a party has committed to some message, $\mathcal{S}$ does not know which message it is. Therefore, we want to be able to make trapdoor commitments and later open them up. Second, when a corrupt party sends a commitment, then $\mathcal{S}$ needs to input some message to $\mathcal{F}_{\mathrm{COM}}^{1:N}$. In this case, we therefore need to have an extractable commitment to the message. The tag-based simulation-extractable commitments presented in Section 5.2 come close to fitting this description.

Our idea is to use each of the n common random strings output by $\mathcal{F}_{\mathrm{MCRS}}$ as a public key for such a commitment scheme. This gives us a set of n commitment schemes, of which at least $t = \lceil \frac{n+1}{2} \rceil$ are secure. Without loss of generality, we will from now on assume we have exactly t secure commitments. In the ideal process, the simulator gets to pick these keys and can therefore pick them as simulation-extractable keys.

To commit to a message m, a party makes a (t, n)-threshold secret sharing of it and commits to each secret share using a different commitment scheme. When making a trapdoor commitment, we make honest commitments to $n-t$ random shares for the adversarial keys, and trapdoor commitments with the simulation-extractable keys. Since the adversary knows at most $n - t < t$ shares, we can later open the commitment to any message we want by fitting the remaining t shares and trapdoor opening the commitments to these shares. To extract a message m, we extract t shares from the simulation-extractable commitments. We can now combine the shares to get the adversarial message.

One remaining issue is when the adversary recycles a commitment or parts of it. This way, we may risk that it uses a trapdoor commitment made by an honest party, in which case we are unable to extract a message. To guard against this problem, we will let the tag for the simulation-extractable commitment scheme contain the identity of the sender P_i, forcing the adversary to use a different tag, which in turn enables us to extract.

Another problem arises, when the adversary corrups a party, which enables it to send messages on behalf of this party. At this point, however, we learn the message so we just need to force it to reuse the same message if it reuses parts of the trapdoor commitment. We therefore introduce a second trapdoor commitment scheme, use this trapdoor commitment scheme to commit to the shares of the message, and insert it in the tag as well. Therefore, if reusing a tag, the adversary must also reuse the same share given by this tag.

Commitment: On input $(\mathbf{vector}, sid, (ck_1, \sigma_1), \ldots, (ck_n, \sigma_n))$ from $\mathcal{F}_{\mathrm{MCRS}}$ and $(\mathrm{commit}, sid, m)$ from $\mathcal{Z}$, the party P_i does the following. He makes a (t, n)-threshold secret sharing $s_1, \ldots, s_n$ of m. He picks randomizers r_j and makes commitments $c_j := \mathrm{Com}_{ck}(s_j; r_j)$. He also picks randomizers R_j and makes tag-based commitments $C_j := \mathrm{Com}_{ck}((P_i, c_j); s_j; R_j)$. The commitment is $c := (c_1, C_1, \ldots, c_n, C_n)$. He broadcasts $(\mathbf{broadcast}, sid, c)$.

Receiving commitment: A party on input $(\mathbf{vector}, sid, (ck_1, \sigma_1), \ldots, (ck_n, \sigma_n))$ from $\mathcal{F}_{\mathrm{MCRS}}$ and $(\mathbf{broadcast}, sid, P_i, c)$ from $\mathcal{F}_{\mathrm{BC}}$ broadcasts $(\mathbf{broadcast}, sid, P_i, c)$.

Once it receives similar broadcasts from all parties, all containing the same P_i, c, it outputs $(\mathbf{commit}, sid, P_i)$ to the environment.

Opening commitment: Party P_i wishing to open the commitment broadcasts $(\mathbf{open}, sid, s_1, r_1, R_1, \ldots, s_n, r_n, R_n)$.

Receiving opening: A party receiving an opening $(\mathbf{open}, sid, P_i, s_1, , r_1, R_1, \ldots, s_n, r_n, R_n)$ from $\mathcal{F}_{\mathrm{BC}}$ to a commitment it earlier received, checks that all commitments are correctly formed $c_j = \mathrm{Com}_{ck_j}(s_j; r_j)$ and $C_j = \mathrm{Com}_{\sigma_j}((P_i, c_j); s_j; r_j)$. It also checks that $s_1, ldots, s_n$ all are valid shares of a (t, n)-threshold secret sharing of some message m. In that case it outputs $(\mathbf{open}, sid, P_i, m)$.

Theorem 7 *The protocol securely realizes $\mathcal{F}_{\mathrm{COM}}^{1:N}$ in the $(\mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\mathrm{MCRS}})$-hybrid model, assuming simulation-extractable commitment schemes exist in the common random string model.*

Sketch of proof. We describe the ideal-process adversary $\mathcal{S}$ and sketch why it is secure along the way. It will run a black-box simulation of $\mathcal{A}$ and what $\mathcal{A}$ sees. In particular, it will simulate the parties $P_1, \ldots, P_N$ and the ideal functionalities $\mathcal{F}_{\mathrm{MCRS}}$ and $\mathcal{F}_{\mathrm{BC}}$. The dummy parties that are actually involved in the protocol and communicate with $\mathcal{Z}$ are written as $\tilde{P}_1, \ldots, \tilde{P}_N$.

Communication: Forward all communication between $\mathcal{A}$ and $\mathcal{Z}$. Also, whenever $\mathcal{A}$ delivers a message to a party P_i, simulate this delivery.

Common random strings: Whenever $\mathcal{A}$ asks $\mathcal{F}_{\mathrm{MCRS}}$ for a common random string, select $(ck, tk) \leftarrow K_{\mathrm{trapdoor}}(1^k)$ and $(\sigma, \tau, \xi) \leftarrow K_{\mathrm{sim-com}}(1^k)$ and return $(\mathbf{crs}, sid, (ck, \sigma))$, while storing $(ck, tk, \sigma, \tau, \xi)$.

When $\mathcal{A}$ inputs $(\mathbf{vector}, sid, (ck_1, \sigma_1), \ldots, (ck_n, \sigma_n))$ to $\mathcal{F}_{\mathrm{MCRS}}$ check that more than half the pairs $(ck_1, \sigma_1), \ldots, (ck_n, \sigma_n)$ match the stored public keys. In that case, send $(\mathbf{vector}, sid, (ck_1, \sigma_1), \ldots, (ck_n, \sigma_n))$ to all parties and halt the simulation of $\mathcal{F}_{\mathrm{MCRS}}$. Note, we only need t stored keys, so if there are more than t honest key pairs, we just act as if we only knew t of the trapdoors.

Commitment by honest party: On receiving $(\mathbf{commit}, sid, P_i)$ from $\mathcal{F}_{\mathrm{COM}}^{1:N}$ we learn that P_i has made a commitment, albeit we do not know the message. We wait until $\mathcal{A}$ has submitted reference strings to $\mathcal{F}_{\mathrm{MCRS}}$ and delivers them to P_i.

We select a (t, n)-threshold secret sharing $s_1, \ldots, s_n$ of 0. For the $n - t$ reference strings where we do not know the keys, including the ones where we do not know the secret keys, we commit to s_j as $c_j := \mathrm{Com}_{tk}(s_j; r_j)$ and $C_j := \mathrm{Com}_{\sigma_j}(P_i, c_j; s_j; R_j)$. For the remaining t reference strings, we make trapdoor commitments $(c_j, ek_j) \leftarrow \mathrm{Tcom}(tk)$ and $(C_j, EK_j) \leftarrow \mathrm{Topen}_{\tau_j}(P_i, c_j)$. We simulate broadcasting $(\mathbf{broadcast}, sid, c_1, C_1, \ldots, c_n, C_n)$.

The process for receiving a commitment is exactly the same as in the protocol, when simulated parties see the commitments they broadcast it. When everybody has broadcast, they are supposed to output $(\mathbf{commit}, sid, P_i)$ to the environment. $\mathcal{S}$ therefore delivers the corresponding commitment message from $\mathcal{F}_{\mathrm{COM}}^{1:N}$ to the dummy party.

Opening: When $\mathcal{S}$ receives $(\mathbf{open}, sid, P_i, m)$ from $\mathcal{F}_{\mathrm{COM}}^{1:N}$ it means that $\tilde{P}_i$ has been instructed to open the commitment, and it was a commitment to m. We recall the $n - t$ shares that we committed to honestly, and fit them into a (t, n)-threshold secret sharing $s_1, \ldots, s_n$ of m. We open the $n - t$ commitments c_j, C_j correctly. We then trapdoor open the t commitments c_j, C_j where we know the corresponding equivocation keys as $r_j \leftarrow \mathrm{Topen}_{ek_j}(s_j)$ and $R_j \leftarrow \mathrm{Topen}_{EK_j}((P_i, c_j), s_j)$. We broadcast $(\mathbf{broadcast}, sid, s_1, r_1, R_1, \ldots, s_n, r_n, R_n)$.

Receiving an opening: On receiving an opening of an earlier received commitment, we check that the commitments contains a consistent (t, n)-threshold secret sharing of $s_1, \ldots, s_n$ of a message m and for all j we have $c_j = \mathrm{Com}_{ck}(s_j; r_j)$ and $C_j = \mathrm{Com}_{\sigma_j}(P_i, c_j; s_j; R_j)$. In that case, we deliver $(\mathbf{open}, sid, P_i, m)$ from $\mathcal{F}_{\mathrm{COM}}^{1:N}$ to our dummy party that outputs the opening to $\mathcal{Z}$.

Commitment by corrupt party: When a corrupt party makes a commitment $(c_1, C_1, \ldots, c_n, C_n)$ with a valid signature so our simulated party would output $(\mathbf{commit}, sid, P_i)$, we need to input some message to $\mathcal{F}_{\mathrm{COM}}^{1:N}$ so we can make the correspodning dummy party output this in the ideal process.

We use the extraction keys, to extract t committed values $s_j \leftarrow \mathrm{Extract}_{\xi_j}((P_i, c_j), C_i)$. The only case, where we cannot do this is when the tag (P_i, c_j) has been used before by P_i, because then it may be a trapdoor commitment we are looking at. However, this can only happen if P_i used (P_i, c_j) as a tag when it was honest, and then upon corruption we have made a trapdoor opening of c_j to some s_j and therefore do not need to do any extraction.

We then reconstruct m from these shares and input $(\mathbf{commit}, sid, m)$ to $\mathcal{F}_{\mathrm{COM}}^{1:N}$ on behalf of the dummy party. In case we did not manage to extract a message, we input $m := 0$ to $\mathcal{F}_{\mathrm{COM}}^{1:N}$, which is ok as long as we do not end up in a situation, where we need to ask $\mathcal{F}_{\mathrm{COM}}^{1:N}$ to open the commitment. This causes $\mathcal{F}_{\mathrm{COM}}^{1:N}$ to send out $(\mathbf{commit}, sid, P_i)$ messages to all dummy parties that we can deliver when needed in the simulation.

Opening by corrupt party: When a corrupt party wants to open a commitment, we check the opening and if acceptable we input $(\mathbf{open}, sid)$ to $\mathcal{F}_{\mathrm{COM}}^{1:N}$. If any honest party receives the opening, we deliver the message $(\mathbf{open}, sid, P_i, m)$ to the corresponding dummy party $\tilde{P}_j$ that outputs it to the environment.

Corruption: In case a party P_i is corrupted, we corrupt the corresponding dummy party $\tilde{P}_i$. We need to simulate the history of this party. If the party has not yet made a commitment, this is easy since there is no history to simulate. If the party has already opened the commitment, we just need to reveal the randomness used in generating the one-time signature.

If the party has made a commitment but not yet opened it, we must simulate an opening of it. On corrupting $\tilde{P}_i$, we learn the message it committed to, so we can use the opening simulation for honest parties described earlier.

To see that this gives us a good simulation, consider the following hybrid experiments for adversary $\mathcal{A}$ and environment $\mathcal{Z}$.

Hybrid 1: This is the protocol executed with $\mathcal{A}$ and environment $\mathcal{Z}$.

Hybrid 2: This is the protocol, where we store $(ck, tk, \sigma, \tau, \xi)$ and return (ck, σ), whenever $\mathcal{A}$ queries $\mathcal{F}_{\mathrm{MCRS}}$ for a common reference string.

Since both commitment scheme have pseudorandom keys, hybrid 1 and 2 cannot be distinguished.

Hybrid 3: This is hybrid 2 modified such that honest party P_i for t commitments where it knows the key, creates equivocal commitments using the trapdoor keys, instead of making real commitments. To produce the openings, it then uses the equivocation keys to generate randomizers so the commitments open to the relevant shares.

Hybrid 2 and hybrid 3 are indistinguishable due to the trapdoor properties of the commitment schemes.

Hybrid 4: We modify hybrid 3 such that when an honest party P_i makes a commitment, it uses a (t, n)-threshold secret sharing of 0 instead of a threshold secret sharing of m. In the opening phase, it opens the $n - t$ pairs (c_j, C_j) where it does not know the trapdoors honestly to the s_j it committed to. It reconstructs shares s_j for the t equivocal commitments so $s_1, \ldots, s_n$ is a (t, n)-threshold secret sharing of m. It then opens the equivocal commitments to these values.

Hybrid 3 and hybrid 4 are perfectly indistinguishable, since $n - t < t$ shares in a (t, n)-threshold secret sharing scheme do not reveal anything about m.

Hybrid 5: We now turn to modify the way we handle corrupt parties. Whenever a corrupt party P_i submits a commitment $(c_1, C_1, \ldots, c_n, C_n)$ to $\mathcal{F}_{\mathrm{BC}}$, we want to extract a message.

For any of the t C_j's where we know the key, there are two cases to consider. One case is where (P_i, c_j) has been used as a tag when P_i was still honest. In this case, we learned an opening s_j, r_j of

c_j upon corruption, and will therefore consider s_j the share. The second case is when (P_i, c_j) has not been used as a tag in a simulation-extractable commitment. In that case, we can extract a share s_j.

We now have t shares, so we can recombine them to get a possible message m. We input $(\mathbf{commit}, sid, m)$ on behalf of $\tilde{P}_i$. In case anything fails, we input $m := 0$ on behalf of $\tilde{P}_i$.

Hybrid 4 and hybrid 5 are indistinguishable. The problem arises if the extracted m does not match the opening. There are two ways this could happen. One possibility is that c_j created by an honest party that is later corrupted is opened to a different share than in the simulation. However, this would imply a breach of the binding property of the commitment scheme. Another possibility is that the extraction fails. However, this would imply breaking the simulation-extractability of the commitment scheme.

We conclude the proof by observing that hybrid 5 is identical to the simulation.

5.5 Coin-Flipping

We will now generate a common random string. The parties will the following natural coin-flipping protocol.

Commitment: P_i chooses at random $r_i \leftarrow \{0,1\}^{\ell(k)}$. It submits $(\mathbf{commit}, sid, r_i)$ to $\mathcal{F}_{\text{COM}}^{1:N}$. $\mathcal{F}_{\text{COM}}^{1:N}$ on this input sends $(\mathbf{commit}, sid, P_i)$ to all parties.

Opening: Once P_i sees $(\mathbf{commit}, sid, ssid, P_j)$ for all j, it sends $(\mathbf{open}, sid, ssid, r_i)$ to $\mathcal{F}_{\text{COM}}^{1:N}$. $\mathcal{F}_{\text{COM}}^{1:N}$ on this input sends $(\mathbf{open}, sid, ssid, P_i, r_i)$ to all parties.

Output: Once P_i sees $(\mathbf{commit}, sid, ssid, P_j, r_j)$ for all j, it outputs $(\mathbf{crs}, sid, \oplus_{j=1}^{N} r_j)$ and halts.

Theorem 8 *The protocol securely realizes (perfectly) the ideal common reference string generator $\mathcal{F}_{\text{CRS}}$ in the $\mathcal{F}_{\text{COM}}^{1:N}$ hybrid model.*

Proof. Consider the following ideal process adversary $\mathcal{S}$ working in the $\mathcal{F}_{\text{CRS}}$-hybrid model, giving it a common reference string σ. It runs a simulated copy of $\mathcal{A}$, a simulated copy of $\mathcal{F}_{\text{COM}}^{1:N}$ and simulated parties $P_1, \ldots, P_N$, not to be confused with the dummy parties $\tilde{P}_1, \ldots, \tilde{P}_N$ that interact with $\mathcal{Z}$ and $\mathcal{F}_{\text{CRS}}$. Whenever $\mathcal{A}$ communicates with the environment $\mathcal{Z}$ it simply forwards those messages. We now list the events that can happen in the protocol.

On activation of P_i, it simulates $\mathcal{F}_{\text{COM}}^{1:N}$ receiving a commitment from P_i by outputting $(\mathbf{commit}, sid, P_i)$ to all parties and $\mathcal{A}$.

On delivery of commitments from all parties to an honest party P_i, it selects r_i at random, subject to the continued satisfiability of condition $\sigma = \oplus_{j=1}^{N} r_j$ and stores it. It then simulates $\mathcal{F}_{\text{COM}}^{1:N}$ receiving an opening of P_i's commitment to r_i.

In case $\mathcal{A}$ corrupts a party P_i, we corrupt the corresponding dummy party $\tilde{P}_i$. If P_i has made a commitment but it has not yet been opened, we select r_i at random, subject to the continued satisfiability of the condition $\sigma = \oplus_{j=1}^{N} r_j$, and simulate that this was the commitment P_i made. In all other cases of corruption, either r_i has not yet been selected, or the commitment has already been opened and $\mathcal{A}$ already knows r_i. The two experiments, $\mathcal{A}$ running with parties $P_1, \ldots, P_N$ in the $\mathcal{F}_{\text{COM}}^{1:N}$-hybrid model, and $\mathcal{S}$ running with dummy parties $\tilde{P}_1, \ldots, \tilde{P}_N$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model are perfectly indistinguishable to $\mathcal{Z}$. To see this, consider a hybrid experiment, where we run the simulation and choose all r_i's at random and then set $\sigma := \oplus_{i=1}^{N} r_i$. Inspection shows that this gives a perfect simulation of $\mathcal{Z}$'s view of the protocol in the $\mathcal{F}_{\text{COM}}^{1:N}$-hybrid model. At the same time, also here we get a uniform random distribution on σ and the r_j's subject to the condition $\sigma = \oplus_{j=1}^{N} r_j$. $\square$

5.6 Multi-party Computation

We are now ready to prove that any well-formed ideal functionality can be securely realized in the multi-string model.

Theorem 9 *For any well-formed functionality $\mathcal{F}$ there is a non-trivial protocol that securely realizes it in the $(\mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\mathrm{MCRS}})$-hybrid model, provided enhanced trapdoor permutations, augmented non-committing encryption and dense cryptosystems exists.*

Proof. Canetti et al. [CLOS02] show that assuming the existence of enhanced trapdoor permitation, dense cryptosystems and augmented non-committing encryption, there is a non-trivial protocol that securely realizes $\mathcal{F}$ in the $(\mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\mathrm{CRS}})$-hybrid model.

Theorem 8 shows that we can securely realize $\mathcal{F}_{\mathrm{CRS}}$ in the $\mathcal{F}_{\mathrm{COM}}^{1:N}$-hybrid model. Therefore, by the universal composability theorem [Can01], we can securely realize $\mathcal{F}$ in the $(\mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\mathrm{COM}}^{1:N})$-hybrid model.

Theorem 7 shows that we can securely realize $\mathcal{F}_{\mathrm{COM}}^{1:N}$ in the $(\mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\mathrm{MCRS}})$-hybrid model assuming the existence of extractable trapdoor commitments. Recall from Theorem 6 that dense cryptosystems imply the existence of extractable trapdoor commitments. By the universal composability theorem we get that $\mathcal{F}$ can be securely realized in the $(\mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\mathrm{MCRS}})$-hybrid model under these assumptions. $\square$

References

[Adl78] Leonard M. Adleman. Two theorems on random polynomial time. In *proceedings of FOCS '78*, pages 75–83, 1978.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *proceedings of CRYPTO '04, LNCS series, volume 3152*, pages 41–55, 2004.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *proceedings of FOCS '04*, pages 186–195, 2004.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *proceedings of STOC '88*, pages 103–112, 1988.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *proceedings of FOCS '01*, pages 136–145, 2001. Full paper available at `http://eprint.iacr.org/2000/067`.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *proceedings of CRYPTO '01, LNCS series, volume 2139*, pages 19–40, 2001. Full paper available at `http://eprint.iacr.org/2001/055`.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *proceedings of STOC '02*, pages 494–503, 2002. Full paper available at `http://eprint.iacr.org/2002/140`.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *proceedings of CRYPTO '02, LNCS series, volume 2442*, pages 581–596, 2002. Full paper available at `http://www.brics.dk/RS/01/41/index.html`.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *proceedings of STOC '89*, pages 25–32, 1989.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, 2005.

[GMY03] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In *proceedings of EUROCRYPT '03, LNCS series, volume 2656*, pages 177–194, 2003. Full paper available at `http://eprint.iacr.org/2003/037`.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.

[GOS06a] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for nizk. In *proceedings of CRYPTO '06, LNCS series, volume 4117*, pages 97–111, 2006.

[GOS06b] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero-knowledge for np. In *proceedings of EUROCRYPT '06, LNCS series, volume 4004*, pages 339–358, 2006.

[Gro06] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *proceedings of ASIACRYPT '06, LNCS series*, 2006. Full paper available at `http://www.brics.dk/~jg/NIZKGroupSignFull.pdf`.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computation*, 28(4):1364–1396, 1999.

[MY04] Philip D. MacKenzie and Ke Yang. On simulation-sound trapdoor commitments. In *proceedings of EUROCRYPT '04, LNCS series, volume 3027*, pages 382–400, 2004. Full paper available at `http://eprint.iacr.org/2003/252`.