Using Level-1 Homomorphic Encryption To Improve Threshold DSA Signatures For Bitcoin Wallet Security

Dan Boneh[1], Rosario Gennaro[2], and Steven Goldfeder[3]

[1] Stanford University
dabo@cs.stanford.edu
[2] City College, City University of New York
rosario@cs.ccny.cuny.edu
[3] Princeton University
stevenag@cs.princeton.edu

Abstract. Recently Gennaro et al. (ACNS '16) presented a threshold-optimal signature algorithm for DSA. Threshold-optimality means that if security is set so that it is required to have $t + 1$ servers to cooperate to sign, then it is sufficient to have $n = t + 1$ honest servers in the network. Obviously threshold optimality compromises robustness since if $n = t + 1$, a single corrupted player can prevent the group from signing. Still, in their protocol, up to t corrupted players cannot produce valid signatures. Their protocol requires six rounds which is already an improvement over the eight rounds of the classic threshold DSA of Gennaro et al. (Eurocrypt '99) (which is not threshold optimal since $n \geq 3t + 1$ if robust and $n \geq 2t + 1$ if not).

We present a new and improved threshold-optimal DSA signature scheme, which cuts the round complexity to **four** rounds. Our protocol is based on the observation that given an encryption of the secret key, the encryption of a DSA signature can be computed in only four rounds if using a level-1 Fully Homomorphic Encryption scheme (i.e. a scheme that supports at least one multiplication), and we instantiate it with the very efficient level-1 FHE scheme of Catalano and Fiore (CCS '15).

As noted in Gennaro et al. (ACNS '16), the schemes have very compelling application in securing Bitcoin wallets from thefts happening due to DSA secret key exposure. Given that network latency can be a major bottleneck in an interactive protocol, a scheme with reduced round complexity is highly desirable. We implement and benchmark our scheme and find it to be very efficient in practice.

1 Introduction

In a threshold signature scheme, the ability to sign a message is shared among n servers such that any group of size $t+1$ can sign, but t or less servers cannot. The immediate consequence is that if a network adversary corrupts up to t servers (we call this a *t-adversary*), it will still not be able to sign a message.

A scheme is said to be *threshold-optimal* if we can set $n = t + 1$ and still prevent a t-adversary from forging signatures. Note that if one sets $n = t + 1$, a t-adversary can always mount a *denial-of-service* attack by refusing to cooperate, leaving the lone honest server unable to sign (actually for $n = t + 1$ even a 1-adversary can mount such an attack). A scheme is said to be *robust* if a t-adversary cannot prevent the group from signing. Clearly robustness requires $n \geq 2t + 1$ so that there always are at least $t + 1$ honest servers in the network.

Much research has been devoted to building secure and efficient threshold signature protocols for a variety of signature schemes. Threshold signatures provide increased security in the presence of break-ins that can compromise one's secret key. By sharing the key among n servers, a user forces the adversary to compromise many of them (a security level parametrized by t) possibly even in a short period of time (by using so-called *proactive* schemes [14]).

Threshold DSA. Threshold scheme for the Digital Signature Algorithm (DSA) were presented in many works. We focus on the protocols described in [26, 27, 35, 36]. The "classic" protocol in [26, 27] requires 8 communication rounds to complete and also $n \geq 3t + 1$ if desiring robustness, and $n \geq 2t + 1$ without. Note that the protocol is not threshold-optimal.

In particular this protocol rules out the simple $n = 2, t = 1$ case (e.g. 2-out-of-2, where two servers have to cooperate to sign). That case was discussed and solved in [36] using techniques that inform much of the work in [25] and this paper. Lindell recently presented an improved version of [36] that achieves much faster speeds by eliminating many of the costly zero-knowledge proofs, but that protocol is also exclusively for the 2-party case, whereas we are interested in the general case.

The Threshold DSA Scheme in [25]. Gennaro et al. solve the above problem by presenting a threshold DSA scheme which is threshold optimal, with a constant (6) number of rounds, and constant local long-term storage [25].

The main idea of the paper comes from [36]: to use a threshold cryptosystem to provide players with shares of the DSA secret key. A threshold cryptosystem achieves the same notion of threshold security but for encryption rather than signatures. The ability to decrypt a ciphertext is shared among n servers in such a way that any group of size $t + 1$ can decrypt, but t or less servers cannot. If the servers are provided with $\alpha = E(x)$ for such a threshold cryptosystem E, then this implicitly constitutes a secret sharing of the value x.

Following [36], Gennaro et al. show that if $x = sk_{DSA}$ (i.e. the secret key of a DSA signature scheme), and E is an additively homomorphic threshold encryption scheme[4] then we can build a threshold DSA scheme with the above properties.

As shown by the implementation presented in [25] their scheme is reasonably practical and efficient, and the response from the Bitcoin community to the work

[4] i.e. a scheme where given $c = E(m)$ and $c' = E(m')$ it is possible to compute $\hat{c} = E(m + m')$ where $+$ is a group operation over the message space, e.g. [41] and its threshold version in [31].

was overwhelmingly positive. It seems however that the limiting factor in the deployment of the [25] protocol is its round complexity, since network latency is a big problem in practice that was not considered by [25]. This leaves open the question if a better (ideally non-interactive) protocol can be found.

1.1 Our contribution

We present a new threshold signature protocol for DSA with only four rounds of interaction. We make several improvements to the security proof of the best previous scheme, and also answer a previously open question about non-malleable commitments. In particular:

- We present a new threshold-optimal DSA scheme with only *four (4)* rounds, whereas the best protocol until now used six (6) rounds. The reduction in the number of rounds will reduce the slowdown caused by network latency.
- We achieve a better reduction than the proof of [25], and thus our proof enables the use shorter keys in practice.
- The proof of [25] requires, among other things, the use of *independent commitments* [28], but we are able to reduce it to the more standard notion of *non-malleable commitments* [20, 22]. In the process, we prove a result of general interest – we answer the question that was left open by [28] and show that *non-malleability* does indeed imply *independence*. We show that they are equivalent, which may be useful in future works as the independence definition is often easier to work with when writing security proofs.
- We implement our signature generation scheme and benchmark our results. We find that our scheme is more parallelizable than that of [25] and will have better runtimes in practice for sufficiently large threshold sets.
- Aside from the implementation of our own scheme, we also provide the only public implementation of the L1FHE scheme of Catalano and Fiore [16]. We built our software modularly so that the two components are completely decoupled, and the FHE software is fully re-usable for other applications.

While the scheme of [25] has received much positive press in the Bitcoin community, it has yet to be adopted by any commercial Bitcoin wallet. Our scheme is less interactive and less complex to code (as we significantly reduce the number of zero knowledge proofs). We therefore believe that it will be our scheme and not the one from [25] that will be adopted in practice.

1.2 Our solution in a nutshell

The protocol in [25] starts by encrypting the secret key x of the DSA scheme with an additively homomorphic encryption scheme E. If the matching decryption key is shared using a threshold cryptosystem for E then this is a secret sharing of x and the protocols in [25, 36] show how to leverage this to obtain a threshold DSA signature.

Our starting observation is that if one uses a threshold *fully homomorphic encryption* scheme then *any* signature scheme can be turned into a *non-interactive* threshold one. Conceptually the idea is simple: if $\alpha = \texttt{FHE}(x)$ is an encryption of the secret key for a signature scheme, then by using the homomorphic properties of FHE the parties are able to locally compute

$E(\sigma)$ where σ is the signature on any message M. This idea can be seen actually as a special case of the non-interactive multiparty computation protocol based on threshold FHE in [39].

We then turned to optimize the above idea for the specific case of DSA. Recall how (a generic form of) DSA works – given a cyclic group $\mathcal{G}$ of prime order q generated by an element g, a hash function H defined from arbitrary strings into Z_q, and another hash function H' defined from $\mathcal{G}$ to Z_q we define:

- Secret Key x chosen uniformly at random in Z_q.
- Public Key $y = g^x$ computed in $\mathcal{G}$.
- Signing Algorithm on input an arbitrary message M, we compute $m = H(M) \in Z_q$. Then the signer chooses k uniformly at random in Z_q and computes $R = g^k$ in $\mathcal{G}$ and $r = H'(R) \in Z_q$. Then she computes $s = k^{-1}(m + xr) \bmod q$. The signature on M is the pair (r, s).
- Verification Algorithm On input $M, (r, s)$ and y, the receiver checks that $r, s \in Z_q$ and computes

$$R' = g^{ms^{-1} \bmod q} y^{rs^{-1} \bmod q} \text{ in } \mathcal{G}$$

and accepts if $H'(R') = r$.

A straightforward application of the above FHE-based approach would result in a relatively inefficient protocol due to the current state of affairs for FHE. Indeed we can see that a circuit computing a DSA signatures from encryptions of x and k is quite deep since it must compute $R = g^k$ and k^{-1}.

We use the same techniques as [25] for the computation of R: basically each player reveals a "share" of R together with a zero-knowledge proof of its correctness (that can be checked against the encrypted values).

Where we diverge from [25] is in the computation of k^{-1} and s. We assume that our encryption scheme E is level-1 HE. This means that given $E(x)$ it is possible to compute $E(F(x))$ for any function F that can be expressed by an arithmetic circuit of multiplicative depth 1. In other words one can perform an unlimited number of additions over encrypted values but only 1 multiplication.

Given $c_k = E(k)$ for such an encryption E, the parties use a variation of Beaver's inversion protocol [4]. First they generate an encryption $c_\rho = E(\rho)$ for a random value ρ, then using the level-1 property they compute $c_{\rho k} = E(\rho k)$ and decrypt it using the threshold decryption property. Now they have a public value $\eta = \rho k$ which reveals no information about k, but allows them to compute an encryption of k^{-1} as follows. First compute η^{-1} and then use the additive homomorphism to compute $c_{k^{-1}} = \eta^{-1} \times c_\rho = E(\eta^{-1}\rho) = E(k^{-1})$.

Efficient Level-1 HE Instantiation. There are various possible choices to instantiate the Level-1 HE in our protocol. We chose to use the construction

recently presented in [16] where it is shown that any additively homomorphic encryption scheme can be turned into a level-1 HE with small computational overhead. By implementing the underlying additively homomorphic encryption with Paillier's scheme [41] we are able to "recycle" all the other components of the [25] protocol: (i) all the zero-knowledge proofs that show some type of consistency property for public values vs. encrypted values and (ii) the threshold encryption based on [31].

1.3 Improvements to the proof of [25].

A tighter reduction. Aside from the improvements in the protocol, our security proof is also significantly improved. In the proof of [25], if there exists an adversary $\mathcal{A}$ that forges with probability $O\epsilon$ in the centralized DSA scheme, then we can build a forger $\mathcal{F}$ that succeeds with probability $O\epsilon^3$ in the threshold scheme. In terms of concrete security, this means that in order to get an equivalent level of security, one would have to use keys three times as long when using the threshold version of the scheme.

While our simulation of the distributed key generation protocol maintains the $O\epsilon^3$ probability, we get a tighter reduction for the threshold signature generation protocol: $O(\epsilon^2)$. This leads to a smaller key size in practice when we use our threshold signature protocol together with a centralized dealer. We note that this use case is very practical in the Bitcoin application: when one wants to add threshold security to an existing Bitcoin address (i.e. an address that was generated using the centralized DSA key generation scheme), one will deal shares of the existing secret to multiple servers only use the threshold protocol for subsequent signature generation.

Non-malleable vs. Independent Commitments. The proof of [25] relies on independent commitments [28] rather than the more standard notion of non-malleable commitments [22, 20]. While [28] showed that independence implies non-malleability, the converse was hitherto unknown. Thus, the use of independence of their paper was a stronger assumption than the use of non-malleability.

We improved their proof by using non-malleable commitments instead of independent commitments, but in the process we were able to prove a more general and interesting lemma: that independence implies non-malleability, and that the two notions are therefore equivalent.

1.4 Results of implementation

We implemented the Level-1 FHE scheme from [16] as well as our signature generation protocol. We also optimized the code from [25], and achieve much better runtimes than reported in that paper.

We found that the runtime of our protocol was comparable but slightly slower than that of [25]. However, when we parallelized the verification of the zero-knowledge proofs of both protocols and ran it on a four-core machine, we found that our protocol was more parallelizable and outperformed that of [25] for thresholds greater than or equal to 13.

Moreover, we argue that the raw-computation time metric only tells a partial story as it does not account for network latency. In a real network setting, the slightly slower runtime of the serial version of our protocol will be amply compensated by the network savings due to the reduction of rounds.

1.5 Motivation: Bitcoin Wallet Security

Following [25] we now present the main motivation of our work: distributed signing for Bitcoin transactions. Bitcoin is the most widely used electronic currency. In Bitcoin users are identified by *addresses* which can be thought simply as DSA public keys[5]. A user with address pk_1 transfers Bitcoins to another user with address pk_2 simply by digitally signing a statement to that extent using sk_1. Consensus on who owns what is achieved via a distributed public ledger (with which we are not going to concern ourselves). We focus on the issue that a user's Bitcoins are as secure as the secret key of its address. If the secret key is compromised, the adversary can easily steal all the Bitcoins associated with the matching public key by simply transferring the coins to itself.

The suggestion to use the classic threshold DSA protocol in [26, 27] to achieve transparent splitting of signature keys, was rejected by Bitcoin practitioners for various reasons. First of all the protocol in [26, 27] was considered to be too computationally heavy, particularly in the number of rounds. Even more seriously, the lack of threshold optimality would force a user to put online a high number of signing servers (e.g. for a security threshold of $t = 3$ a user would have to deploy $n = 7$ or $n = 10$ servers depending if robustness is required or not) with a substantial operating cost, and higher number of possible infection targets. As discussed in [25] it was quite clear the the Bitcoin community would much prefer a threshold optimal scheme with $n = t + 1$ even if that meant compromising robustness and allowing denial-of-service attacks[6].

While the protocol in [36] achieves threshold-optimality, it is limited to the case of $(n = 2, t = 1)$ and does not allow for flexible[7] choices of n, t.

Bitcoin does have a built-in *multisignature* feature. See Appendix A for an explanation of this feature and why it does not suffice for our application.

[5] Bitcoin uses ECDSA, the DSA scheme implemented over a group of points of an elliptic curve. As in [25] we ignore this fact since our results hold for a generic version of DSA which is independent of the underlying group where the scheme is implemented (provided the group is of prime order and DSA is obviously unforgeable in this implementation.)

[6] The rationale for that is that provided a bad server in a denial-of-service attack can be easily identified – that is the case in both our protocol and the protocol of [25] – then the corrupted server can be rebooted, restarted from a trusted basis, and the adversary eliminated.

[7] A preliminary version of [25] provided a simple extension of [36] to the n-out-of-n case which however required $O(n)$ rounds to complete. The same version also uses a standard combinatorial construction to go from n-out-of-n to the generic n, t case, but that requires $O(n^t)$ local long-term storage by each server.

2 Model, Definitions and Tools

In this section we introduce our communication model and provide definitions of secure threshold signature schemes.

COMMUNICATION MODEL. We assume that our computation model is composed of a set of n *players* $P_1, \ldots, P_n$ connected by a complete network of point-to-point channels and a broadcast channel.

THE ADVERSARY. We assume that an adversary, $\mathcal{A}$, can corrupt up to t of the n players in the network. $\mathcal{A}$ learns all the information stored at the corrupted nodes, and hears all broadcasted messages. We consider two type of adversaries:

- *honest-but-curious:* the corrupted players follow the protocol but try to learn information about secret values;
- *malicious:* corrupted players to divert from the specified protocol in *any* (possibly malicious) way.

We assume the network is "partially synchronous", meaning the adversary speaks last in every communication round (also known as a *rushing* adversary.) The adversary is modeled by a probabilistic polynomial time Turing machine.

Adversaries can also be categorized as *static* or *adaptive*. A static adversary chooses the corrupted players at the beginning of the protocol, while an adaptive one chooses them during the computation. In the following, for simplicity, we assume the adversary to be static, though the techniques from [15, 32] can be used to extend our result to the adaptive adversary case.

Given a protocol $\mathcal{P}$, the *view* of the adversary, denoted by $\mathcal{VIEW}_{\mathcal{A}}(\mathcal{P})$, is defined as the probability distribution (induced by the random coins of the players) on the knowledge of the adversary, namely, the computational and memory history of all the corrupted players, and the public communications and output of the protocol.

SIGNATURE SCHEME A signature scheme $\mathcal{S}$ is a triple of efficient randomized algorithms (Key-Gen, Sig, Ver). Key-Gen is the *key generator* algorithm: on input the security parameter 1^λ, it outputs a pair (y, x), such that y is the *public key* and x is the *secret key* of the signature scheme. Sig is the *signing* algorithm: on input a message m and the secret key x, it outputs *sig*, a signature of the message m. Since Sig can be a randomized algorithm there might be several valid signatures *sig* of a message m under the key x; with $\mathsf{Sig}(m, x)$ we will denote the set of such signatures. Ver is the *verification* algorithm. On input a message m, the public key y, and a string *sig*, it checks whether *sig* is a proper signature of m, i.e. if $sig \in \mathsf{Sig}(m, x)$.

The notion of security for signature schemes was formally defined in [29] in various flavors. The following definition captures the strongest of these notions: existential unforgeability against adaptively chosen message attack.

Definition 1. *We say that a signature scheme* $\mathcal{S}$ =(Key-Gen,Sig,Ver) *is* unforgeable *if no adversary who is given the public key* y *generated by* Key-Gen*, and the*

signatures of k messages $m_1, \ldots, m_k$ adaptively chosen, can produce the signature on a new message m (i.e., $m \notin \{m_1, \ldots, m_k\}$) with non-negligible (in λ) probability.

THRESHOLD SECRET SHARING. Given a secret value x we say that the values $(x_1, \ldots, x_n)$ constitute a (t, n)-threshold secret sharing of x if t (or less) of these values reveal no information about x, and if there is an efficient algorithm that outputs x having $t+1$ of the values x_i as inputs.

THRESHOLD SIGNATURE SCHEMES. Let $\mathcal{S}$=(Key-Gen, Sig, Ver) be a signature scheme. A (t, n)-threshold signature scheme $\mathcal{TS}$ for $\mathcal{S}$ is a pair of protocols (Thresh-Key-Gen, Thresh-Sig) for the set of players $P_1, \ldots, P_n$.

Thresh-Key-Gen is a distributed key generation protocol used to jointly generate a pair (y, x) of public/private keys on input a security parameter 1^λ. At the end of the protocol, the private output of P_i is a value x_i such that the values $(x_1, \ldots, x_n)$ form a (t, n)-threshold secret sharing of x. The public output of the protocol contains the public key y. Public/private key pairs (y, x) are produced by Thresh-Key-Gen with the same probability distribution as if they were generated by the Key-Gen protocol of the regular signature scheme $\mathcal{S}$. In some cases it is acceptable to have a *centralized* key generation protocol, in which a trusted dealer runs Key-Gen to obtain (x, y) and the shares x among the n players.

Thresh-Sig is the distributed signature protocol. The private input of P_i is the value x_i. The public inputs consist of a message m and the public key y. The output of the protocol is a value $sig \in \mathsf{Sig}(m, x)$.

The verification algorithm for a threshold signature scheme is, therefore, the same as in the regular centralized signature scheme $\mathcal{S}$.

Definition 2. *We say that a (t, n)-threshold signature scheme $\mathcal{TS}$ =(*Thresh-Key-Gen,Thresh-Sig*) is* unforgeable*, if no malicious adversary who corrupts at most t players can produce, with non-negligible (in λ) probability, the signature on any new (i.e., previously unsigned) message m, given the view of the protocol* Thresh-Key-Gen *and of the protocol* Thresh-Sig *on input messages $m_1, \ldots, m_k$ which the adversary adaptively chose.*

This is analogous to the notion of existential unforgeability under chosen message attack as defined by Goldwasser, Micali, and Rivest [29]. Notice that now the adversary does not just see the signatures of k messages adaptively chosen, but also the internal state of the corrupted players and the public communication of the protocols. Following [29] one can also define weaker notions of unforgeability.

In order to prove unforgeability, we use the concept of *simulatable adversary view* [13, 30]. Intuitively, this means that the adversary who sees all the information of the corrupted players and the signature of m, could generate by itself all the other public information produced by the protocol Thresh-Sig. This ensures that the run of the protocol provides no useful information to the adversary other than the final signature on m.

Definition 3. *A threshold signature scheme $\mathcal{TS}$ =(*Thresh-Key-Gen,Thresh-Sig*) is* simulatable *if the following properties hold:*

1. *The protocol* Thresh-Key-Gen *is simulatable. That is, there exists a simulator* SIM_1 *that, on input a public key* y*, can simulate the view of the adversary on an execution of* Thresh-Key-Gen *that results in* y *as the public output.*
2. *The protocol* Thresh-Sig *is simulatable. That is, there exists a simulator* SIM_2 *that, on input the public input of* Thresh-Sig *(in particular the public key* y *and the message* m*),* t *shares* $x_{i_1}, \ldots, x_{i_t}$*, and a signature* sig *of* m*, can simulate the view of the adversary on an execution of* Thresh-Sig *that generates* sig *as an output.*

Threshold Optimality. As in [25], we are interested in a *threshold-optimal* scheme. Given a (t, n)-threshold signature scheme, obviously $t+1$ honest players are necessary to generate signatures. A scheme is *threshold-optimal* if $t+1$ honest players also suffice [25].

If we consider an honest-but-curious adversary, then it will suffice to have $n = t + 1$ players in the network to generate signatures (since all players will behave honestly, even the corrupted ones). But in the presence of a malicious adversary one needs at least $n = 2t + 1$ players in total to guarantee *robustness*, i.e. the ability to generate signatures even in the presence of malicious faults. But as we discussed in the introduction, we want to minimize the number of servers, and keep it at $n = t + 1$ even in the presence of malicious faults. In this case we give up on robustness, meaning that we cannot guarantee anymore that signatures will be provided. But we can still prove that our scheme is unforgeable. In other words an adversary that corrupts almost all the players in the network can only create a denial-of-service attack, but not forge signatures.

2.1 Level-1 Homomorphic Encryption

We now define the notion of a Level-1 Homomorphic Encryption scheme. An encryption scheme E defined as usual by a key generation, encryption and decryption algorithms is Level-1 Homomorphic if the following conditions hold:

- The message space is integers modulo a given (large) integer N;
- The ciphertext space $\mathcal{C}$ is partitioned into two disjoint sets $\mathcal{C}_0, \mathcal{C}_1$. We say that a ciphertext that belongs to $\mathcal{C}_i$ is a ciphertext at *level* i;
- The encryption algorithm for E always outputs ciphertexts at level 0;
- There is an efficiently computable operation $+_E$ over the ciphertext space such that, if $\alpha = Enc(a), \beta = Enc(b) \in \mathcal{C}_i$, where $a, b \in Z_N$, then

$$\gamma = \alpha +_E \beta = E(a + b \bmod N) \in \mathcal{C}_i$$

- There is an efficiently computable operation $\times_E$ over the ciphertext space such that, if $\alpha = Enc(a), \beta = Enc(b) \in \mathcal{C}_0$, where $a, b \in Z_N$, then

$$\gamma = \alpha \times_E \beta = E(ab \bmod N) \in \mathcal{C}_1$$

Instantiation. We use the level-1 homomorphic encryption scheme from [16] which is built in a "black-box" manner from any additively homomorphic encryption scheme (i.e. a scheme for which only the $+_E$ operation exists). For

the latter we follow [25] and use Paillier's encryption scheme (described below). This choice allows us to use unchanged many of the other components of the [25] protocol: (i) all the zero-knowledge proofs that show some type of consistency property for public values vs. encrypted values and (ii) the threshold Paillier's cryptosystem from [31].

Paillier's Cryptosystem.

- Key Generation: generate two large primes P, Q of equal length. and set $N = PQ$. Let $\lambda(N) = lcm(P-1, Q-1)$ be the Carmichael function of N. Finally choose $\Gamma \in Z^*_{N^2}$ such that its order is a multiple of N. The public key is (N, Γ) and the secret key is $\lambda(N)$.
- Encryption: to encrypt a message $m \in Z_N$, select $x \in_R Z^*_N$ and return $c = \Gamma^m x^N \bmod N^2$.
- Decryption: to decrypt a ciphertext $c \in Z_{N^2}$, let L be a function defined over the set $\{u \in Z_{N^2} : u = 1 \bmod N\}$ computed as $L(u) = (u-1)/N$. Then the decryption of c is computed as $L(c^{\lambda(N)})/L(\Gamma^{\lambda(N)}) \bmod N$.
- Homomorphic Properties: Given two ciphertexts $c_1, c_2 \in Z_{N^2}$ define $c_1 +_E c_2 = c_1 c_2 \bmod N^2$. If $c_i = E(m_i)$ then $c_1 +_E c_2 = E(m_1 + m_2 \bmod N)$. Similarly, given a ciphertext $c = E(m) \in Z_{N^2}$ and a number $a \in Z_n$ we have that $a \times_E c = c^a \bmod N^2 = E(am \bmod N)$.

Catalano-Fiore Level 1 Homomorphic Encryption. Here we briefly recall the level-1 homomorphic encryption in [16]. Let E be any additively homomorphic encryption scheme. Level-0 ciphertexts are then constructed as follows

$$\mathsf{Enc}(m) = [m - b, E(b)] \quad \text{for} \quad b \in_R Z_N$$

Obviously, component-wise addition of these ciphertexts results in the encryption of the addition of the messages. If $[\alpha_i, \beta_i] = \mathsf{Enc}(m_i)$ then

$$[\alpha_1, \beta_1] +_{\mathsf{Enc}} [\alpha_2, \beta_2] = [\alpha_1 + \alpha_2 \bmod N, \beta_1 +_E \beta_2] = \mathsf{Enc}(m_1 + m_2 \bmod N)$$

Level-1 ciphertexts are created by the multiplication homomorphism

$$[\alpha_1, \beta_1] \times_{\mathsf{Enc}} [\alpha_2, \beta_2] = [\alpha, \beta_1, \beta_2]$$

where

$$\alpha = E(\alpha_1 \alpha_2 \bmod N) +_E \alpha_2 \odot_E \beta_1 +_E \alpha_1 \odot_E \beta_2$$

where with $\odot_E$ we denote the operation of multiplication of a ciphertext by a scalar: if ψ is an integer and $\beta = E(b)$ is a ciphertext, then

$$\psi \odot_E \beta = E(\psi b \bmod N)$$

Addition of level-1 ciphertexts is done by using the $+_E$ operator on the first component and concatenating all the other components (notice that addition of level-1 ciphertexts is not length-preserving).

$$[\alpha, \beta_1, \beta_2] +_{\mathsf{Enc}} [\hat{\alpha}, \hat{\beta}_1, \hat{\beta}_2] = [\alpha +_E \hat{\alpha}, \beta_1, \beta_2, \hat{\beta}_1, \hat{\beta}_2]$$

2.2 Threshold Cryptosystems

In a (t, n)-threshold cryptosystem, there is a public key pk with a matching secret key sk which is shared among n players with a (t, n)-secret sharing. When a message m is encrypted under pk, $t+1$ players can decrypt it via a communication protocol that does not expose the secret key.

More formally, a public key cryptosystem $\mathcal{E}$ is defined by three efficient algorithms:

- key generation $\mathsf{Enc\text{-}Key\text{-}Gen}$ that takes as input a security parameter λ, and outputs a public key pk and a secret key sk.
- An encryption algorithm Enc that takes as input the public key pk and a message m, and outputs a ciphertext c. Since Enc is a randomized algorithm, there will be several valid encryptions of a message m under the key pk; with $\mathsf{Enc}(m, pk)$ we will denote the set of such ciphertexts.
- and a decryption algorithm Dec which is run on input c, sk and outputs m, such that $c \in \mathsf{Enc}(m, pk)$.

We say that $\mathcal{E}$ is semantically secure if for any two messages m_0, m_1 we have that the probability distributions $\mathsf{Enc}(m_0)$ and $\mathsf{Enc}(m_1)$ are computationally indistinguishable.

A (t, n) threshold cryptosystem $\mathcal{TE}$, consists of the following protocols for n players $P_1, \ldots, P_n$.

- A key generation protocol $\mathsf{TEnc\text{-}Key\text{-}Gen}$ that takes as input a security parameter λ, and the parameter t, n, and it outputs a public key pk and a vector of secret keys $(sk_1, \ldots, sk_n)$ where sk_i is private to player P_i. This protocol could be obtained by having a trusted party run $\mathsf{Enc\text{-}Key\text{-}Gen}$ and sharing sk among the players.
- A threshold decryption protocol TDec, which is run on public input a ciphertext c and private input the share sk_i. The output is m, such that $c \in \mathsf{Enc}(m, pk)$.

We point out that threshold variations of Paillier's scheme have been presented in the literature [2, 18, 19, 31]. In order to instantiate our dealerless protocol, we use the scheme from [31] as it includes a dealerless key generation protocol that does not require $n \geq 2t + 1$.

2.3 Non-Malleable Trapdoor Commitments

Trapdoor Commitments. A trapdoor commitment scheme allows a sender to commit to a message with information-theoretic privacy. i.e., given the transcript of the commitment phase, the receiver, even with infinite computing power, cannot guess the committed message better than at random. On the other hand when it comes to opening the message, the sender is only computationally bound to the committed message. Indeed the scheme admits a *trapdoor* whose knowledge allows opening a commitment in any possible way (we will refer to this as *equivocate* the commitment). The trapdoor should be hard to compute efficiently.

Formally a (non-interactive) trapdoor commitment scheme consists of four algorithms KG, Com, Ver, Equiv with the following properties:

- KG is the key generation algorithm, on input the security parameter it outputs a pair pk, tk where pk is the public key associated with the commitment scheme, and tk is called the *trapdoor*.
- Com is the commitment algorithm. On input pk and a message M it outputs $[C(M), D(M)] = \mathsf{Com}(\mathsf{pk}, M, R)$ where r are the coin tosses. $C(M)$ is the commitment string, while $D(M)$ is the decommitment string which is kept secret until opening time.
- Ver is the verification algorithm. On input C, D and pk it either outputs a message M or $\perp$.
- Equiv is the algorithm that opens a commitment in any possible way given the trapdoor information. It takes as input pk, strings M, R with $[C(M), D(M)] = \mathsf{Com}(\mathsf{pk}, M, R)$, a message $M' \neq M$ and a string T. If $T = \mathsf{tk}$ then Equiv outputs D' such that $\mathsf{Ver}(\mathsf{pk}, C(M), D') = M'$.

We note that if the sender refuses to open a commitment we can set $D = \perp$ and $\mathsf{Ver}(\mathsf{pk}, C, \perp) = \perp$. Trapdoor commitments must satisfy the following properties

Correctness If $[C(M), D(M)] = \mathsf{Com}(\mathsf{pk}, M, R)$ then $\mathsf{Ver}(\mathsf{pk}, C(M), D(M)) = M$.

Information Theoretic Security For every message pair M, M' the distributions $C(M)$ and $C(M')$ are statistically close.

Secure Binding We say that an adversary $\mathcal{A}$ wins if it outputs C, D, D' such that $\mathsf{Ver}(\mathsf{pk}, C, D) = M$, $\mathsf{Ver}(\mathsf{pk}, C, D') = M'$ and $M \neq M'$. We require that for all efficient algorithms $\mathcal{A}$, the probability that $\mathcal{A}$ wins is negligible in the security parameter.

Non-malleable Trapdoor Commitments. To define non-malleability [22], think of the following game. The adversary, after seeing a tuple of commitments produced by honest parties, outputs his own tuple of committed values. At this point the honest parties decommit their values and now the adversary tries to decommit his values in a way that his messages are related to the honest parties' ones[8]. Intuitively, we say that a commitment scheme is non-malleable if the adversary fails at this game.

However the adversary could succeed by pure chance, or because he has some a priori information on the distribution of the messages committed by the

[8] We are considering *non-malleability with respect to opening* [20] in which the adversary is allowed to see the decommitted values, and is required to produce a related decommitment. A stronger security definition *(non-malleability with respect to commitment*) simply requires that the adversary cannot produce a commitment to a related message after being given just the committed values of the honest parties. However for information-theoretic commitments (like the ones considered in this paper) the latter definition does not make sense. Indeed information-theoretic secrecy implies that given a commitment, *any* message could be a potential decommitment. What specifies the meaning of the commitment is a valid opening of it.

honest parties. So when we formally define non-malleability for commitments we need to focus on ruling out that the adversary receives any help from seeing the committed values. This can be achieved by comparing the behavior of the adversary in the above game, to the one of an adversary in a game in which the honest parties' messages are not committed to and the adversary must try to output related messages without any information about them.

We now give the formal definition of non-malleability from [17]. We have a publicly known distribution $\mathcal{M}$ on the message space and a randomly chosen public key pk (chosen according to the distribution induced by KG).

Define Game 1 (the real game) as follows. We think of the adversary $\mathcal{A}$ as two separate efficient algorithms $\mathcal{A}_1, \mathcal{A}_2$. We choose t messages according to the distribution $\mathcal{M}$, compute the corresponding commitments and feed them to the adversary $\mathcal{A}_1$. The adversary $\mathcal{A}_1$ outputs a vector of u commitments, with the only restriction that he cannot copy any of the commitments presented to him. $\mathcal{A}_1$ also transfers some internal state to $\mathcal{A}_2$. We now open our commitments and run $\mathcal{A}_2$, who will open the u commitments prepared by $\mathcal{A}$ (if $\mathcal{A}_2$ refuses to open some commitment we replace the opening with $\perp$). We then invoke a distinguisher $\mathcal{D}$ on the two vectors of messages. $\mathcal{D}$ will decide if the two vectors are related or not (i.e. $\mathcal{D}$ outputs 1 if the messages are indeed related). We denote with $\mathsf{Succ1}_{\mathcal{D},\mathcal{A},\mathcal{M}}$ the probability that $\mathcal{D}$ outputs 1 in this game, i.e.

$$\mathsf{Succ1}_{\mathcal{D},\mathcal{A},\mathcal{M}}(k) = Prob\left[\begin{array}{c} \mathsf{pk},\mathsf{tk} \leftarrow \mathsf{KG}(1^k)\ ;\ m_1,\ldots,m_t \leftarrow \mathcal{M}\ ; \\ r_1,\ldots,r_t \leftarrow \{0,1\}^k\ ;\ [c_i,d_i] \leftarrow \mathsf{Com}(\mathsf{pk},m_i,r_i)\ ; \\ (\omega,\hat{c}_1,\ldots,\hat{c}_u) \leftarrow \mathcal{A}_1(\mathsf{pk},c_1,\ldots,c_t) \text{ with } \hat{c}_j \neq c_i \forall i,j\ ; \\ (\hat{d}_1,\ldots,\hat{d}_u) \leftarrow \mathcal{A}_2(\mathsf{pk},\omega,d_1,\ldots,d_t)\ ; \\ \hat{m}_i \leftarrow \mathsf{Ver}(\mathsf{pk},\hat{c}_i,\hat{d}_i)\ : \\ \mathcal{D}(m_1,\ldots,m_t,\hat{m}_1,\ldots,\hat{m}_u) = 1 \end{array}\right]$$

Define now Game 2 as follows. We still select t messages according to $\mathcal{M}$ but this time feed nothing to the adversary $\mathcal{A}$. The adversary now has to come up with u messages on its own. Again we feed the two vectors of messages to $\mathcal{D}$ and look at the output. We denote with $\mathsf{Succ2}_{\mathcal{D},\mathcal{A}}$ the probability that $\mathcal{D}$ outputs 1 in this game, i.e.

$$\mathsf{Succ2}_{\mathcal{D},\mathcal{A},\mathcal{M}}(k) = Prob\left[\begin{array}{c} \mathsf{pk},\mathsf{tk} \leftarrow \mathsf{KG}(1^k)\ ;\ m_1,\ldots,m_t \leftarrow \mathcal{M}\ ; \\ (\hat{m}_1,\ldots,\hat{m}_u) \leftarrow \mathcal{A}(\mathsf{pk})\ ; \\ \text{s.t. } \hat{m}_i \in \mathcal{M} \cup \{\perp\}\ : \\ \mathcal{D}(m_1,\ldots,m_t,\hat{m}_1,\ldots,\hat{m}_u) = 1 \end{array}\right]$$

Finally we say that a distinguisher $\mathcal{D}$ is admissible, if for any input $(m_1,\ldots,m_t, \hat{m}_1,\ldots,\hat{m}_u)$, its probability of outputting 1 does not increase if we change any message $\hat{m}_i$ into $\perp$. This prevents the adversary from artificially "winning" the game by refusing to open its commitments.

We say that the commitment scheme is (t,u) ϵ-non-malleable if for every message space distribution $\mathcal{M}$, every efficient admissible distinguisher $\mathcal{D}$, every $0 < \epsilon < 1$, and for every efficient adversary $\mathcal{A}$, there is an efficient adversary $\mathcal{A}'$

(whose running time is polynomial in ϵ^{-1}) such that

$$|\mathsf{Succ1}_{\mathcal{D},\mathcal{A},\mathcal{M}}(k) - \mathsf{Succ2}_{\mathcal{D},\mathcal{A}',\mathcal{M}}(k)| \leq \epsilon$$

In other words $\mathcal{A}'$ fares almost as well as $\mathcal{A}$ in outputting related messages.

2.4 Independent Trapdoor Commitments

Following [25], our proof uses *independent* commitments as introduced in [28]. Consider the following scenario: an honest party produces a commitment C and the adversary, after seeing C, will produce another commitment C' (which we to require to be different from C in order to prevent the adversary from simply copying the behavior of the honest party and outputting an identical committed value). At this point the value committed by the adversary should be *fixed*, i.e. no matter how the honest party opens his commitment, the adversary will always open in a unique way.

A formal definition is presented below. In [28] the authors proved that independence implies non-malleability, therefore establishing it as a stronger property. Here, however, we show that the two notions are actually equivalent, and therefore we show that both our proof and the one in [25] hold under the more standard (and natural) definition of non-malleability.

The following definition takes into account that the adversary may see and output many commitments ([17]).

Independence For any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ the following probability is negligible in k:

$$Prob\left[\begin{array}{c} \mathsf{pk},\mathsf{tk} \leftarrow \mathsf{KG}(1^k)\ ;\ m_1,\ldots,m_t \leftarrow \mathcal{M} \\ r_1,\ldots,r_t \leftarrow \{0,1\}^k\ ;\ [c_i,d_i] \leftarrow \mathsf{Com}(\mathsf{pk},m_i,r_i) \\ (\omega,\hat{c}_1,\ldots,\hat{c}_u) \leftarrow \mathcal{A}_1(\mathsf{pk},c_1,\ldots,c_t) \text{ with } \hat{c}_j \neq c_i \forall i,j \\ m'_1,\ldots,m'_t \leftarrow \mathcal{M}\ ;\ d'_i \leftarrow \mathsf{Equiv}(\mathsf{pk},\mathsf{tk},m_i,r_i,m'_i) \\ (\hat{d}_1,\ldots,\hat{d}_u) \leftarrow \mathcal{A}_2(\mathsf{pk},\omega,d_1,\ldots,d_t) \\ (\hat{d}'_1,\ldots,\hat{d}'_u) \leftarrow \mathcal{A}_2(\mathsf{pk},\omega,d'_1,\ldots,d'_t) \\ \exists i : \perp \neq \hat{m}_i = \mathsf{Ver}(\mathsf{pk},\hat{m}_i,\hat{c}_i,\hat{d}_i) \neq \mathsf{Ver}(\mathsf{pk},\hat{m}'_i,\hat{c}_i,\hat{d}'_i) = \hat{m}'_i \neq \perp \end{array}\right]$$

In other words even if the honest parties open their commitments in different ways using the trapdoor, the adversary cannot change the way he opens his commitments $\hat{C}_j$ based on the honest parties' opening.

Equivalence of Non-malleability and Independence. In [28] it was shown that independence implies non-malleability, therefore establishing it as a stronger property. Here, however, we show that the two notions are actually equivalent, and therefore we show that both our proof and the one in [25] hold under the more standard (and natural) definition of non-malleability.

Lemma 1. *Let* KG, Com, Ver, Equiv *be a non-malleable commitment. Then* KG, Com, Ver, Equiv *s also an independent commitment.*

Proof. Assume by contradiction that the commitment is not independent, that is there exists an adversary $\mathcal{A}$ that is able to open its commitment in different ways depending on the opening of its input commitments. Then construct the following distinguisher $\mathcal{D}$ in the definition of non-malleable commitments: given the messages $m_1, \ldots, m_t$ and the messages $\mu_1, \ldots, \mu_u$ output by $\mathcal{A}$, we have that $\mathcal{D}(m_1, \ldots, m_t, \mu_1, \ldots, \mu_u) = 1$ if the μ_i are indeed the messages that $\mathcal{A}$ reveals when the m_i are opened. So $\mathcal{D}$ always outputs 1 with adversary $\mathcal{A}$.

Now another adversary $\mathcal{A}'$ who does *not* see any information about the m_i except for the message distribution $\mathcal{M}$, will only be able to guess the correct μ_i with probability substantially bounded away from 1 (the probability that m_i appears as the message tuple in $\mathcal{M}$, which is definitely bounded away from 1 by a non-negligible quantity – at least the probability that m'_i is selected).

2.5 Candidate Non-Malleable/Independent Trapdoor Commitments

The non-malleable commitment schemes in [20, 21] are not suitable for our purpose because they are not "concurrently" secure, in the sense that the security definition holds only for $t = 1$ (i.e. the adversary sees only 1 commitment).

The stronger concurrent security notion of non-malleability for $t > 1$ is achieved by the schemes presented in [17, 24, 37]). Therefore for the purpose of our threshold DSA scheme, we can use any of the schemes in [17, 24, 37]).

3 The new scheme

We start by giving an informal description of the initialization phase and the key generation protocol which are identical to the ones in [25]. Readers are referred to [25] for details. We will then get into the details of our new signature generation protocol and how it differs from the one in [25].

Initialization phase. As in [25] a common reference string containing the public information pk for a non-malleable[9] trapdoor commitment KG, Com, Ver, Equiv is selected and published. This could be accomplished by a trusted third party, who can be assumed to erase any secret information (i.e. the trapdoor of the commitment) after selection or via some publicly verifiable method that generates the public information, without the trapdoor being known.

Key Generation. The parties run the key generation protocol from [31] to generate a public key E for the Paillier's encryption scheme, together with a sharing of its matching secret key. The value N for Paillier's scheme is chosen such that

$N > q^8$. Then as in [25] a value x is generated, and encrypted with E, with the value $\alpha = E(x)$ made public.This is an implicit (t, n) secret sharing of x, since the decryption key of E is shared among the players. We use non-malleable

[9] In [25] they require an independent commitment scheme, but following our Lemma1 it suffices that the scheme is non-malleable.

commitments KG, Com, Ver, Equiv to enforce the independence of the values contributed by each player to the selection of x. Note that the resulting distribution of public DSA keys generated by the protocol is not necessarily uniform, but as proven in [25] it has sufficiently high entropy to guarantee unforgeability.

3.1 Signature Generation

We now describe our new signature generation protocol, which is run on input m (the hash of the message M being signed).

In the following with $\bigoplus_{i=1}^{t+1} \alpha_i$ we denote the summation over the addition operation $+_E$ of the encryption scheme: i.e. $\bigoplus_{i=1}^{t+1} \alpha_i = \alpha_1 +_E \ldots +_E \alpha_{t+1}$. Similarly with $\odot_E$ we denote the operation of multiplication of a ciphertext by a scalar: if ψ is an integer and $\alpha = E(a)$ is a ciphertext, then

$$\psi \odot_E \alpha = \bigoplus_{i=1}^{\psi} \alpha = E(\psi a \bmod N)$$

Moreover in the protocol below we assume that if any commitment opens to $\perp$ or if any of the ZK proofs fails, the protocol outputs $\perp$.

- $\mathsf{Round\ 1}$
  Each player P_i
  - chooses $\rho_i, k_i \in_R Z_q$ and $c_i \in_R [-q^6, q^6]$
  - computes $r_i = g^{k_i}$
  - computes $u_i = E(\rho_i)$, $v_i = E(k_i)$ and $w_i = E(c_i)$
  - computes $[C_i, D_i] = \mathsf{Com}([r_i, u_i, v_i, w_i])$ and broadcasts C_i
- $\mathsf{Round\ 2}$
  Each player P_i broadcasts
  - D_i. This allows everybody to compute $[r_i, u_i, v_i, w_i] = \mathsf{Ver}(C_i, D_i)$
  - a zero-knowledge argument $\Pi_{(i)}$ which states
    $\exists \nu_1, \nu_2 \in [-q^3, q^3]$ and $\nu_3 \in [-q^6, q^6]$:
    * $g^{\nu_1} = r_i$
    * $D(v_i) = \nu_1$
    * $D(u_i) = \nu_2$
    * $D(w_i) = \nu_3$
- $\mathsf{Round\ 3}$
  Each player P_i
  - verifies the ZKPs of all other players
  - computes $R = \Pi_1^{t+1} r_i = g^k$ and $r = H'(R) \in Z_q$
  - computes $u = \bigoplus_{i=1}^{t+1} u_i = E(\rho)$, $v = \bigoplus_{i=1}^{t+1} v_i = E(k)$ and $w = \bigoplus_{i=1}^{t+1} w_i = E(c)$ where $\rho = \sum_{i=1}^{t+1} \rho_i$, $k = \sum_{i=1}^{t+1} k_i$ and $c = \sum_{i=1}^{t+1} c_i$ (all over the integers)
  - computes $z = E(k\rho + cq) = (v \times_E u) +_E (q \odot_E w)$
  - jointly decrypt z using TDec to learn the value $\eta = D(z) \bmod q = k\rho \bmod q$

– Round 4
  Each player P_i
  - computes $\psi = \eta^{-1} \bmod q$
  - computes $\hat{v} = E(k^{-1}) = \psi \odot_E u$
  - computes

$$\begin{aligned}\sigma &= \hat{v} \times_E [(E(m) +_E (r \odot \alpha)] \\ &= E(k^{-1}(m + xr)) \\ &= E(s)\end{aligned}$$

  The players invoke distributed decryption protocol TDec over the ciphertext σ. Let $s = D(\sigma) \bmod q$. The players output (r, s) as the signature for m.

The size of the modulus N. We note that since $N > q^8$ all the plaintext operations induced by the ciphertext operations $+_E, \times_E, \odot_E$ are over the integers. In turns this implies that the reduction modulo q are correct.

The Zero-knowledge arguments. The ZK arguments invoked by our protocol are nearly identical to the ones in [25] due to the fact that we are using Paillier's scheme to implement our level-1 homomorphic encryption (and we only require a subset of the proofs needed in that protocol). As in [25, 36] the proofs require an auxiliary RSA modulus $\tilde{N}$ to construct the "range proofs" via [23]. Moreover the security of the arguments require the strong RSA assumption on the modulus $\tilde{N}$. For details readers are referred to [25, 36].

Our protocol only requires zero-knowledge proofs on level-0 ciphertexts. Since the proofs that we use are a subset of the ones used in [25], we can recycle their proofs with one simple modification. Recall that when we instantiate the cryptosystem of [16] using Paillier as the underlying scheme, the level-0 ciphertexts are of the form

$$\mathsf{Enc}(m) = [m - b, E(b)] \quad \text{for} \quad b \in_R Z_N$$

where $E(b)$ is a Paillier encryption of b. In order to directly utilize the proofs of [25], however, we need $E(m)$, a Paillier encryption of m. We can obtain a Paillier encryption of m by deterministically encrypting $m - b$, and computing the sum of these ciphertexts using the addition operator of Paillier. In particular, let $E(b) = \Gamma^b x^N \bmod N^2$ for some $x \in Z_N^*$. Then

$$E(m) = E(b) \times [\Gamma^{m-b} 1^N \bmod N^2] \bmod N^2 = \Gamma^m x^N \bmod N^2$$

The prover now has an encryption of m and can use the zero knowledge proofs from [25] directly. Moreover, because the encryption of $m - b$ is deterministic, the verifier can perform the operation himself, and thus be convinced that the ciphertexts being used for the proofs is a Pallier encryption of the same value as the original level-0 ciphertext.

The homomorphism of the encryption E. Note that the scheme performs two multiplications of ciphertexts, but in each case the ciphertexts are of level 0. So level-1 homomorphism suffices.

4 Security Proof

We prove the following Theorem:

Theorem 1. *Assuming that*

- *The DSA signature scheme is unforgeable;*
- *E is a semantically secure, additively homomorphic encryption scheme;*
- KG, Com, Ver, Equiv *is a non-malleable trapdoor commitment;*
- *the Strong RSA Assumption holds;*

then our threshold DSA scheme in the previous section is unforgeable.

The proof follows from a standard simulation: if there is an adversary $\mathcal{A}$ that forges in the threshold scheme with a significant probability, then there exists a forger $\mathcal{F}$ that forges in the centralized DSA scheme also with a significant probability. The adversary $\mathcal{A}$ will be run in a simulated environment by the forger $\mathcal{F}$ which will use the forgery produced by $\mathcal{A}$ as its own forgery.

The forger $\mathcal{F}$ runs on input a public key y for DSA. Its first task is to run a simulation for $\mathcal{A}$ that terminates with y as the public key of the threshold signature scheme. We refer to [25] for a simulation of the key generation protocol[10].

As discussed in [25] we cannot simulate an exact distribution for the public keys generated by the key generation protocol, but we can only generate keys at random over a sufficiently large subset of all possible keys. This is still enough to prove unforgeability (in other words our $\mathcal{F}$ will only work on a polynomially large fraction of public keys which is still a contradiction to the unforgeability of DSA). For those subset of keys, the view of the adversary during the simulated protocol is indistinguishable from its view during a real execution.

Now whenever $\mathcal{A}$ requests the signature of a message m_i, the forger $\mathcal{F}$ can obtain the real signature (r_i, s_i) from its signature oracle. It will then simulate an execution of the threshold signature protocol which is indistinguishable from the real one (in particular on input m_i it will output $\perp$ or a correct signature with essentially the same probability as in the real case – when the protocol terminates with a signature, the output will be (r_i, s_i).).

Because these simulations are indistinguishable from the real protocol for $\mathcal{A}$, the adversary will output a forgery with the same probability as in real life. Such a forgery m, r, s is a signature on a message that was never queried by $\mathcal{F}$ to its signature oracle and therefore a valid forgery for $\mathcal{F}$ as well.

We now present some more details about the simulation of the signature generation protocol.

4.1 Signature generation simulation

During this simulation the forger $\mathcal{F}$ will handle signature queries issued by the adversary $\mathcal{A}$. We recall that during the simulation we assume that

[10] Again, in [25] the proof requires independent commitments but thanks to our Lemma 1 we can relax that assumption to non-malleable commitments.

- $\mathcal{F}$ controls the lone honest player, and that without loss of generality this player is P_1 and it always speaks first at each round;
- $\mathcal{F}$ can equivocate any of the commitment produced by P_1 during the simulation, since $\mathcal{F}$ sets up the CRS for the adversary during the initialization phase, and can do so with knowledge of the trapdoor for the commitment scheme.

During the simulation $\mathcal{F}$ has access to a signing oracle that produces DSA signatures under the public key $y = g^x$ issued earlier to $\mathcal{F}$. However the ciphertext α that in the real execution contains an encryption of x, during the simulation contains the encryption of a different value τ known to $\mathcal{F}$.

As in the real case in the simulation below, we assume that if any commitment opens to $\perp$ or if any of the ZK proofs fails, the simulation aborts.

When $\mathcal{A}$ requests to sign a message M, such that $m = H(M)$, the forger $\mathcal{F}$ obtains a signature (r, s) from its signature oracle. $\mathcal{F}$ first computes $R = g^{ms^{-1} \bmod q} y^{rs^{-1} \bmod q} \in \mathcal{G}$. Note that $H'(R) = r \in Z_q$ due to the fact that the signature is valid. Also $\mathcal{F}$ chooses a random value $\eta \in_R [-q^7, q^7]$ such that $\eta^{-1}(m + r\tau) = s \bmod q$

The simulation then proceeds as follows:

- Round 1
  Each player P_i
  - chooses $\rho_i, k_i \in_R Z_q$ and $c_i \in_R [-q^6, q^6]$
  - computes $r_i = g^{k_i}$
  - computes $u_i = E(\rho_i)$, $v_i = E(k_i)$ and $w_i = E(c_i)$
  - computes $[C_i, D_i] = \mathsf{Com}([r_i, u_i, v_i, w_i])$ and broadcasts C_i
- Round 2
  Each player P_i broadcasts
  - D_i. This allows everybody to compute $[r_i, u_i, v_i, w_i] = \mathsf{Ver}(C_i, D_i)$
  - the zero-knowledge argument $\Pi_{(i)}$

  At this point $\mathcal{F}$ rewinds the adversary to the beginning of the round and changes the opening of P_1 to $[r'_1, u'_1, v'_1, w'_1]$ such that:
  - $r'_1 = R \cdot \Pi_{j=2}^{t+1} r_j$
  - $u'_1 = E(\rho'_1)$, such that $\rho'_1 + \sum_{i=2}^{t+1} \rho_i = 1$;
  - $v'_1 = E(k'_1)$ such that

$$(k'_1 + \sum_{i=2}^{t+1} k_i) + q \sum_{i=1}^{t+1} c_i = \eta$$

  and simulates the appropriate ZK proof for P_1. If after the rewinding any player P_i changes the opening of its commitment to $D'_i \neq D_i$ then the forger $\mathcal{F}$ **aborts**.
- Round 3
  Each player P_i
  - verifies the ZKPs of all other players
  - computes $R = \Pi_1^{t+1} r_i = g^k$ and $r = H'(R) \in Z_q$

- computes $u = \bigoplus_{i=1}^{t+1} u_i = E(1)$, $v = \bigoplus_{i=1}^{t+1} v_i$ and $w = \bigoplus_{i=1}^{t+1} w_i$
- computes $z = (v \times_E u) +_E (q \odot_E w) = E(\eta)$
- jointly decrypt z using TDec to learn the value η

– Round 4
Each player P_i
- computes $\psi = \eta^{-1} \bmod q$
- computes $\hat{v} = E(\eta^{-1}) = \psi \odot_E u$ since $u = E(1)$
- computes

$$\begin{aligned}\sigma &= \hat{v} \times_E [(E(m) +_E (r \odot \alpha)] \\ &= E(\eta^{-1}(m + r\tau)) \\ &= E(s)\end{aligned}$$

The players invoke distributed decryption protocol TDec over the ciphertext σ. Let $s = D(\sigma) \bmod q$. The players output (r, s) as the signature for m.

Lemma 2. *On any input M the simulation terminates with $\mathcal{F}$ aborting only with negligible probability.*

Proof (of Lemma 2). $\mathcal{F}$ aborts only if the adversary changes its opening of the commitments after the rewinding in Round 2. This is obviously ruled out by the independence property of the commitment scheme that we use in the protocol. More precisely, due to Lemma 1 the non-malleable commitment scheme that we use in the protocol is also independent. The independence property guarantees that the adversary can change its opening only with negligible probability.

Lemma 3. *The simulation terminates in polynomial time and is indistinguishable from the real protocol.*

Proof (of Lemma 3). The only differences between the real and the simulated views are

– in the simulated view the forger $\mathcal{F}$ might abort, but as proven in Lemma 2, this only happens with negligible probability;
– in the simulated view, the plaintexts encrypted in the ciphertexts published by $\mathcal{F}$ do not satisfy the same properties that they would in the protocol when they were produced by a real player P_1. It is not hard to see that in order to distinguish between the two views one must be able to break the semantic security of the encryption scheme.
– $\mathcal{F}$ runs simulated ZK proofs instead of real ones that would prove those properties. But the simulations are statistically indistinguishable from the real proofs.
– The distribution of the value η. In the real protocol, η is a fixed value $k\rho$ (which we know is bounded by q^6 at most because of the ZK proofs), masked by a random value in the range of q^7. In our protocol, η is a random value in the range of q^7. It is not hard to see that the two distributions are statistically indistinguishable.

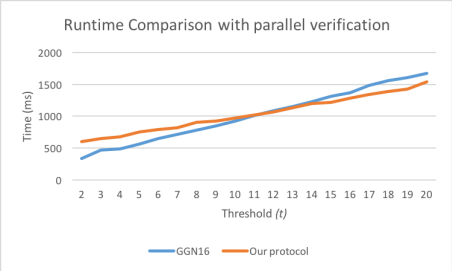

Fig. 1: Runtime comparison between our scheme and that of [25] on a four core machine when we parallelize the ZKP verification.

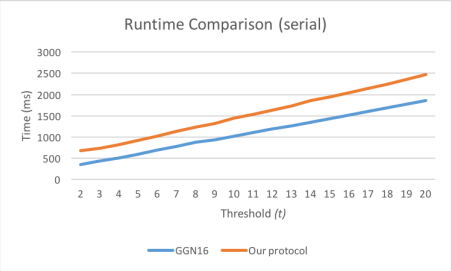

Fig. 2: Runtime comparison between our scheme and that of [25] running on a single core.

4.2 Concrete Analysis

Assuming the our adversary $\mathcal{A}$ forges with probability ϵ, the concrete analysis in [25] shows that $\mathcal{F}$ has a roughly ϵ^3 probability of forging. In [25] this bound applies to both the simulation of the key generation and signature protocol. When simulating both protocols, our proof achieves the same bound, since our simulation of the key generation protocol is basically identical to the one in [25].

However, our simulation of the signature generation protocol is substantially different than the one in [25] since we do not require every single protocol to successfully complete with a correct signature. Indeed in order to guarantee that $\mathcal{F}$ forges, it is not necessary that every single message M queried by the adversary is correctly signed. It is sufficient that the protocol execution on input M is indistinguishable from the real one, even if the input is $\perp$[11]. This results in a better reduction, where the success probability of $\mathcal{F}$ is approximately ϵ^2.

The practical implication is that in application where the key generation does not have to be simulated (e.g. where the key has already been chosen and it is then shared, or the key generation and sharing is done by a trusted party), our proof yields a reduction with better parameters.

5 Implementation Report

We provide an open-source Java implemented of our signature scheme, and compare it to the runtimes of [25]. All benchmarks were done on an Ubuntu desktop with an Intel® quad-core i7-6700 CPU @ 3.40GHz and 64GB of RAM.

We implemented our code in Java to be consistent with the implementation in [25] so that we could get an accurate comparison of the runtime. We re-used the code from their paper when possible, and thus also used the independent trapdoor commitment scheme from [24] using the Jpair library.

[11] This is not possible in the key generation part, since $\mathcal{F}$ must "hit" the target public key y in order to subsequently forge.

We were able to make improvements to the code of [25] that sped up the runtime significantly. Firstly, rather than using Java's built-in BigInteger class for modular exponentiation and inversion, we used Square's jna-gmp[12] instead. In Figure 3, we show the effects of making this switch. Secondly, considering that for sufficiently large thresholds, the signature generation time is dominating by verifying other players' proofs, we added some parallelization support for the zero-knowledge proof verification. In order to make sure that the benchmark comparison was accurate, we made all improvements both to our code as well as to the code from [25].

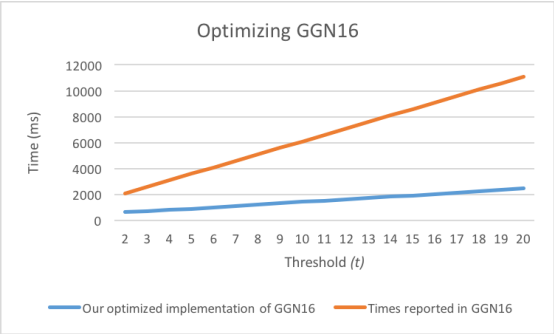

Fig. 3: Comparison between the runtime reported in GGN16 and the runtime that we achieved by using jna-gmp. We note that the benchmark machine was not the same and worse in their case, and thus this speedup is mostly but not entirely due to our optimizations. All benchmarks were done on a single core.

For the underlying Paillier scheme, we modified the Java implementation of threshold Paillier in [42]. We fixed an undocumented overflow error in the library that caused decryption to fail for threshold values greater than or equal to 15.

We did not know of any Java implementation (or any open source implementation) of the Level-1 FHE scheme from [16], so we implemented that as well, and this may be of independent interest.

As is the case with the scheme of [25], the cost of verifying commitments and zero-knowledge proofs will dominate the base proving time for most threshold parameters. As the proof verification contains multiple checks that are highly parallelizable, we added parallelization support to both our implementation as well as that of [25].

In Figure 1, we compare the performance of our scheme to the scheme of [25] for threshold sets of up to size 20. We found that for thresholds of 13 players or more, our scheme outperformed the one of [25] when run on our four-core benchmark machine. In all of these parameter sets, our scheme is highly efficient and finished in under 2 seconds. We stress that the benchmarks only depend on t, the threshold, and not on n, the total number of players in the scheme.

[12] https://github.com/square/jna-gmp

In Figure 2, we compare the runtime of our scheme to that of [25] when we turn off parallelization. The runtimes are comparable, but ours are somewhat slower. It emerges that while our scheme is slower on a single thread, it is more parallelizable. Intuitively, this makes sense as we are condensing the computation into fewer rounds, and thus there is more room for parallelization.

We stress that both Figure 1 and Figure 2 reflect the computation time of a single player, which will be the computation time of the protocol as all players can run in parallel. However these benchmarks do not take network communication time into effect. Even for the serial implementation, in a real network setting, the slight loss of performance of our protocol will be amply compensated for by the reduction of two rounds of communication. This is particularly true when the number of players increases as we cannot proceed to the next round until all players have received the output from every player in the previous round and posted their output for the current round.

6 Acknowledgements

This work was supported by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

Rosario Gennaro is supported by NSF Grant 1565403. Steven Goldfeder is supported by the NSF Graduate Research Fellowship under grant number DGE 1148900 and NSF award CNS-1651938.

References

1. G. Andresen, "Github: Shared Wallets Design," https://gist.github.com/gavinandresen/4039433, accessed: 2014-03-20.
2. O. Baudron, P.-A. Fouque, D. Pointcheval, G. Poupard and J. Stern. *Practical Multi-Candidate Election System.* PODC'01
3. N. Barić, and B. Pfitzmann. *Collision-free accumulators and Fail-stop signature schemes without trees.* Proc. of EUROCRYPT'97 (LNCS 1233), pp.480–494, Springer 1997.
4. Judit Bar-Ilan, Donald Beaver: *Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction.* PODC 1989: 201-209
5. Bitcoin Forum member dree12, "List of Bitcoin Heists," https://bitcointalk.org/index.php?topic=83794.0, 2013.
6. Bitcoin Forum member gmaxwell, "List of Bitcoin Heists," https://bitcointalk.org/index.php?topic=279249.0, 2013.
7. "Bitcoin wiki: Transactions," https://en.bitcoin.it/wiki/Transactions, accessed: 2014-02-11.
8. "Bitcoin wiki: Elliptic Curve Digital Signature Algorithm," https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm, accessed: 2014-02-11.

9. "Bitcoin wiki: Elliptic Curve Digital Signature Algorithm," https://en.bitcoin.it/w/index.php?title=Secp256k1&oldid=51490, accessed: 2014-02-11.
10. J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten :Mixcoin: Anonymity for bitcoin with accountable mixes,' in *Financial Cryptography and Data Security*. Springer, 2014, pp. 486–504.
11. J. Camenisch, A. Kiayias, and M. Yung: On the portability of generalized schnorr proofs. In: Advances in Cryptology-EUROCRYPT 2009, pp. 425–442. Springer (2009)
12. J. Camenisch, S. Krenn, and V. Shoup: A framework for practical universally composable zero-knowledge protocols. In: Advances in Cryptology–ASIACRYPT 2011, pp. 449–467. Springer (2011)
13. R. Canetti. *Universally Composable Security: A new paradigm for cryptographic protocols.* Proc. of 42^{nd} IEEE Symp. on Foundations of Computer Science (FOCS'01), pp.136–145, 2001.
14. R. Canetti, R. Gennaro, A. Herzberg and D. Naor. *Proactive security: Long-term protection against break-ins.* RSA Laboratories CryptoBytes 3 (1), 1-8, 1997.
15. R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin: Adaptive Security for Threshold Cryptosystems. CRYPTO 1999, LNCS Vol.1666, pp 98-115
16. Dario Catalano, Dario Fiore. *Using Linearly-Homomorphic Encryption to Evaluate Degree-2 Functions on Encrypted Data.* ACM Conference on Computer and Communications Security 2015: 1518-1529
17. I. Damgård, J. Groth. *Non-interactive and reusable non-malleable commitment schemes.* Proc. of 35^{th} ACM Symp. on Theory of Computing (STOC'03), pp.426-437, 2003.
18. I. Damgård and M. Jurik. *A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System.* PKC'01, LNCS Vol.1992, pp.119–136
19. I. Damgård, M. Koprowski: Practical Threshold RSA Signatures without a Trusted Dealer. EUROCRYPT 2001: LNCS Vol.2045, pp. 152-165
20. G. Di Crescenzo, Y. Ishai, R. Ostrovsky. Non-Interactive and Non-Malleable Commitment. Proc. of 30^{th} ACM Symp. on Theory of Computing (STOC'98), pp.141–150, 1998.
21. G.Di Crescenzo, J. Katz, R. Ostrovsky, A. Smith. *Efficient and Non-interactive Non-malleable Commitment.* Proc. of EUROCRYPT 2001, Springer LNCS 2045, pp.40-59.
22. D. Dolev, C. Dwork and M. Naor. *Non-malleable Cryptography.* SIAM J. Comp. 30(2):391–437, 200.
23. E. Fujisaki, T. Okamoto: Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. CRYPTO 1997: LNCS Vol.1294, pp.16-30
24. R. Gennaro. *Multi-trapdoor Commitments and Their Applications to Proofs of Knowledge Secure Under Concurrent Man-in-the-Middle Attacks.* Proc. of CRYPTO'04, Springer LNCS 3152, pp.220–236.
25. R. Gennaro, S. Goldfeder, A. Narayanan. *Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security.* ACNS 2016.
26. R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. *Threshold DSS Signatures.* EUROCRYPT'96, LNCS Vol.1070, pp. 354–371.
27. R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. *Secure Distributed Key Generation For Discrete Log Based Cryptosystems.* EUROCRYPT'99, LNCS Vol.1592, pp. 295–310.

28. R. Gennaro and S. Micali. *Independent Zero-Knowledge Sets*. ICALP 2006, LNCS vol.4052, pp. 34–45.
29. S. Goldwasser, S. Micali, and R.L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.
30. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM. J. Computing*, 18(1):186–208, February 1989.
31. C. Hazay, G.L. Mikkelsen, T. Rabin, T. Toft. and A.A. Nicolosi: Efficient RSA key generation and threshold Paillier in the two-party setting.
32. S. Jarecki, A. Lysyanskaya. Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures. EUROCRYPT 2000: LNCS Vol.1807, pp.221-242
33. D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.
34. Kaspersky Labs, "Financial cyber threats in 2013. Part 2: malware," http://securelist.com/analysis/kaspersky-security-bulletin/59414/financial-cyber-threats-in-2013-part-2-malware/, 2013.
35. Y. Lindell. Fast Secure Two-Party ECDSA Signing. *IACR Cryptology ePrint Archive*, 2017:552, 2017.
36. P. MacKenzie and M. Reiter. *Two-party Generation of DSA Signatures*. Int. J. Inf. Secur. (2004)
37. P. MacKenzie and K. Yang. *On Simulation-Sound Commitments.* Proc. of EUROCRYPT'04, Springer LNCS 3027, pp.382-400.
38. S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 127–140.
39. Pratyay Mukherjee, Daniel Wichs: *Two Round Multiparty Computation via Multi-key FHE.* EUROCRYPT (2) 2016: 735-763
40. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Consulted*, vol. 1, p. 2012, 2008.
41. P. Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. EUROCRYPT'99, LNCS Vol.1592, pp. 223-238
42. *Paillier Threshold Encryption Toolbox* http://cs.utdallas.edu/dspl/cgi-bin/pailliertoolbox/manual.pdf
43. T. Pedersen. Distributed provers with applications to undeniable signatures. In D. Davies, editor, *Advances in Cryptology–EUROCRYPT'91*, Lecture Notes in Computer Science Vol. 547, Springer-Verlag, 1991.
44. R. Rivest, A. Shamir and L. Adelman. A Method for Obtaining Digital Signature and Public Key Cryptosystems. *Comm. of ACM*, 21 (1978), pp. 120–126
45. A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612–613, 1979.

A Multisignatures vs. Threshold Signatures

A simple way to achieve the above protection against break-ins for *any* signature scheme, is to give each server P_i in the network an independent secret key sk_i. The public key for the group is the collection $pk = (pk_1, \ldots, pk_n)$ of public keys of each individual players. To sign a message M the group has to produce at least

$t+1$ valid signatures for the appropriate subset of public keys. This approach is called *multisignature schemes*.

Threshold signatures on the other hand are *transparent* in the sense that there is a unique public key pk and signatures are verified against this key. No outside user is aware of how the matching secret key sk is stored by the user: i.e. on how many servers n and with what threshold t.

Bitcoin's protocol allows users to employ multisignatures to split the signing power among many clients. There are obvious efficiency reasons to prefer threshold signatures to multisignatures since the bandwidth and verification cost grow linearly in t in the latter. But as discussed in [25] there is also a more serious side effect to the use of multisignatures. Anonymity In Bitcoin is predicated on the use by a single party of many different random addresses and the hope that addresses belonging to the same users cannot be linked together by looking at the set of transactions in the public ledger. We already know that this a pretty weak notion of anonymity (see e.g. [38]) but multisignatures weakens it further by providing evidence of common ownership for addresses that use the same (n, t) multisignature security policy (see [25] for details). Threshold signatures do not suffer from these problems as the splitting of the key is done completely on the client side. As in [25], our threshold signatures the adversary can introduce some bias into the distribution of the signatures, but it is far superior to multisignatures which leak the entire access policy onto the public blockchain.