# Fast Multiparty Threshold ECDSA with Fast Trustless Setup

Rosario Gennaro[1] and Steven Goldfeder[2]

[1] City University of New York
rosario@cs.ccny.cuny.edu
[2] Cornell Universty[§]
goldfeder@cornell.edu

**Abstract.** A threshold signature scheme enables distributed signing among $n$ players such that any subgroup of size $t+1$ can sign, whereas any group with $t$ or fewer players cannot. While there exist previous threshold schemes for the ECDSA signature scheme, we present the first protocol that supports multiparty signatures for any $t \leq n$ with efficient, dealerless key generation. Our protocol is faster than previous solutions and significantly reduces the communication complexity as well. We prove our scheme secure against malicious adversaries with a dishonest majority. We implemented our protocol, demonstrating its efficiency and suitability to be deployed in practice.

**Note: This revised version fixes some crucial details in the protocol. The proof of the protocol described in the previous version is not correct, though no attack has been shown that exploits the bug in the proof. More details appear in the intoduction.**

## 1 Introduction

A threshold signature scheme enables $n$ parties to share the power to issue digital signatures under a single public key. A threshold $t$ is specified such that any subset of $t+1$ players can jointly sign, but any smaller subset cannot. Generally, the goal is to produce signatures that are compatible with an existing centralized signature scheme. In a threshold scheme the key generation and signature algorithm are replaced by a communication protocol between the parties, but the verification algorithm remains identical to the verification of a signature issued by a centralized party.

In recent years there has been renewed attention to this topic, in particular to the threshold generation of ECDSA signatures, mostly due to the use of ECDSA in Bitcoin and other cryptocurrencies. Indeed, a secure threshold signature schemes for ECDSA would be an effective countermeasure to the constant theft of bitcoins due to the compromise of the secret signing key that authorizes transactions. Securing Bitcoin is equivalent to securing these keys. Instead of storing them in a single location, keys should be split and signing should be authorized by a threshold set of computers. A breach of any one of these machines—or any number of machines less than the threshold—will not allow the attacker to steal any money or glean any information about the key.

Before the advent of Bitcoin, the best ECDSA threshold signature scheme was the work by Gennaro *et al.* [18], which has a considerable setback. To implement a security threshold of $t$ players (i.e. $t$ or less players cannot sign) it is necessary to share the key among at least $2t+1$ players, and the participation of at least $2t+1$ players is required to sign. This limitation rules out an $n$-of-$n$ sharing where all parties are required to sign. Furthermore, it requires setting up many servers holding key shares, which may be costly and also makes the job of the attacker easier in

This is a major revision of our paper [16] that appeared at ACM CCS 2018.
[§] Steven Goldfeder was at Princeton University when this research took place

some way (as there are more servers that can be targeted and while the honest participants need $2t+1$ players to sign, an attacker need only compromise $t+1$ servers to steal the key).

In an attempt to address these issues, Mackenzie and Reiter built a specialized scheme for the 2-out-of-2 signature case (i.e. $t = 1$ and $n = 2$) [30], a case not covered by Gennaro *et al.*'s scheme. Recently much improved 2-out-of-2 schemes have been presented [12, 28]. However 2-out-of-2 sharing is very limited and can't express more flexible sharing policies that might be required in certain applications.

Gennaro and others in [17] (improved in [4]) address the more general $(t, n)$ case in the *threshold optimal* setting, meaning $n \geq t+1$ and that only $t+1$ players are needed to sign. However, their scheme too has a setback in that the distributed key generation protocol is very costly.

**Our Result:** We present a new threshold-optimal protocol for ECDSA that improves in many significant ways over [4, 17]. Our new protocol is faster and requires much less communication than [4, 17]; it is also conceptually simpler and does not require a complicated distributed key generation protocol (details of the comparison appear below).

In concurrent work that appeared in the same proceedings, Lindell *et al.* present a similar protocol for multiparty threshold ECDSA with an efficient key generation [29].

### 1.1 Overview of our solution

Consider a "generic" DSA signature algorithm that works over any cyclic group $\mathcal{G}$ of prime order $q$ generated by an element $g$. It uses a hash function $H$ defined from arbitrary strings into $Z_q$, and another hash function $H'$ defined from $\mathcal{G}$ to $Z_q$. The secret key is $x$ chosen uniformly at random in $Z_q$, with a matching public key $y = g^x$. To sign a message $M$, the signer computes $m = H(M) \in Z_q$, chooses $k$ uniformly at random in $Z_q$ and computes $R = g^{k^{-1}}$ in $\mathcal{G}$ and $r = H'(R) \in Z_q$. Then she computes $s = k(m + xr) \bmod q$. The signature on $M$ is the pair $(r, s)$ which is verified by computing

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

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

The technical complication with sharing DSA signatures comes from having to jointly compute $R$ (which requires raising $g$ to the inverse of a secret value $k$) and to compute $s$ which requires multiplying two secret values $k, x$. As shown in [18], it is sufficient to show how to compute two multiplications over secret values that are shared among the players. In [18] the values are shared via Shamir's secret sharing, i.e., as points on a polynomial of degree $t$ with free term the secret. The effect of multiplication is that the degree of the polynomial is doubled, which explains why the solution in [18] requires at least $2t+1$ players to participate. To address this problem [30] uses a multiplicative sharing of the secret key $x$ as $x = x_1 \cdot x_2$ (an approach taken also in [12, 28]) which is however hard to generalize to $t > 2$.

A different approach was taken in [17]: the secret key $x$ is encrypted under a public key encryption scheme $E$, and it is the secret key of $E$ that is shared among the players, effectively providing a secret sharing of $x$. If $E$ is an additively homomorphic encryption scheme (e.g. Paillier's [33]) they show that it is possible to construct a reasonably efficient protocol, with a few troubling bottlenecks. The major one is that the protocol requires a joint generation of the public key/secret key pair for the additively homomorphic encryption $E$. When $E$ is instantiated using Paillier, this requires the distributed generation of an RSA modulus. Although solutions are known for this problem (e.g. [24]), they are far from scalable and efficient. To our knowledge the protocol from [24] has never been implemented for the malicious multiparty case. The only benchmark we are aware of for this protocol is that for the two-party semi-honest case it takes 15 minutes [28], and we can extrapolate that it would take significantly longer in the multiparty malicious setting. Moreover the signature generation protocols in [4, 17] require long messages and complicated ZK proofs.

In this paper we take a different path inspired by the SPDZ approach to multiparty computation [9]. Given two secrets $a, b$ shared additively among the players, i.e. $a = a_1 + \ldots + a_n$ and $b = b_1 + \ldots + b_n$ where $P_i$ holds $a_i$, and $b_i$, we want to generate an additive sharing of $c = ab$. We note that $ab = \sum_{i,j} a_i b_j$ and therefore to get an additive sharing of $ab$, it is sufficient to obtain an additive sharing of each individual term $a_i b_j$. To that extent we use a 2-party protocol that allows two parties to transform multiplicative shares of a secret to additive shares of the same secret. The players engage in this protocol in a pairwise fashion to obtain an additive sharing of the product $ab$.

Using this approach, we build a simple and elegant threshold ECDSA protocol for the general multiparty setting. The players start with a $(t, n)$ Shamir sharing of the secret key $x$. When $t + 1$ players want to sign, they generate an additive sharing of two random values $k = \sum_i k_i$ and $\gamma = \sum_i \gamma_i$ and they use the above idea to compute additive sharings of the products $\delta = k\gamma$ (which is reconstructed in the clear) and $\sigma = kx = \sum_i w_i$ (which is kept shared). By multiplying the local shares of $\gamma$ by the public value $\delta^{-1}$ the players end up with an additive sharing[3] of $k^{-1}$. The value $R$ is then easily computed in the exponent $R = \prod_i g^{\gamma_i \delta^{-1}}$. The value $s$ is shared additively among the players since each player holds $s_i = k_i m + w_i r$ and $s = \sum_i s_i$.

### 1.2 Avoid expensive ZK Proofs in case of a Malicious Adversary

Following [28] we make minimal use of ZK proofs to detect malicious behavior by the players.

Instead we take an "optimistic" approach and run the protocol assuming everybody is honest. We then check the validity of the resulting signature to detect if there were players who deviated from the protocol (if the signature does not verify then obviously at least one player did not follow the instructions).

At that point, because we possibly have a dishonest majority among the players, there is no guarantee that we can generate a correct signature so the protocol stops and aborts. This creates a technical complication in the proof as we have to make sure that the values revealed by the good players do not leak any valuable information, not only in the case of good executions, but also in the case of aborting executions. As we will see, this will require us to "distributively" check that the shares $s_i$ reconstruct a valid signature before revealing them. This check is somewhat reminiscent of the way Canetti and Goldwasser solve a similar problem in [7] to construct *threshold CCA secure encryption* based on the Cramer-Shoup scheme.

Range Proofs. Even when using the signature verification step to detect cheating, we have to run two relatively expensive ZK proofs during the share conversion protocol:

- a "range proof" that certain values encrypted under Paillier's encryption scheme are "small";
- a proof that a party knows $x$ such that $c = E(x)$ and $y = g^x$ where $E$ is Paillier's encryption scheme.

### 1.3 The issue with the previous version

The previous version of the protocol did not check the range of all values encrypted under Paillier's encryption scheme. This creates an adversarial strategy that could leak information about an honest party's share. The proof of the protocol in the previous version did not take into account this leakage and incorrectly assumed that certain subprotocols were zero-knowledge and simulatable. The issue was reported in [37, 32].

Moreover in [37] the authors show how to leverage this information leakage to actually recover the entire share of an honest party, and for the adversary to learn the entire secret key of the group. Crucially, however, the attack relies on choosing a very small Paillier modulus $N$ (approximately the same size of the modulus $q$ used in the DSA scheme). In our previous version we clearly stated

[3] This is the famous Bar-Ilan and Beaver inversion trick [1].

that $N > q^7$ therefore the attack does not apply to our previous scheme (though it worked against implementations that neglected to check the size of the modulus).

We have now changed the protocol to make sure that all values encrypted under Paillier's scheme are checked to be "small". All changes from the previous protocol are discussed when they arise and marked with **Note about previous version** header.

Finally, in the previous version we discussed a protocol where the ZK range proofs were removed entirely. We discussed how removing these ZK proofs creates an attack that leaks some information about the DSA secret key (and the randomizer $k$ used in each signature) shared among the servers. We conjectured that this information would still not allow the adversary to forge signatures. We did not consider this "light" protocol to be "secure", and never recommended it for implementation in commercial systems. We made the conjecture to stimulate research on the "danger" of leaking that information. Our conjecture has been disproven in [37, 32] where it is also shown that removing the ZK range proofs does indeed lead to an insecure protocol. We are going to leave Section 5 in the paper as an historical record but will preface it with this new information.

### 1.4 Experimental Results

We implemented our scheme and found both the key generation and signing protocols to be very efficient.

The key generation protocol is easy to implement and is quite fast (under a second for any reasonable choice of parameters). This is in stark contrast to [4, 17] for which the key generation protocol has never been implemented, and it is hard to estimate what the actual running time would be.

Our signing protocol is also extremely efficient, and is a significant improvement over previous works both in terms of data transferred and running time.

With the combination of an efficient key generation and signing protocol, our scheme is suitable to be deployed in practice. We present full benchmarks and evaluations in Section 7.

## 2 Preliminaries

Communication Model. We assume the existence of a broadcast channel as well as point-to-point channels connecting every pair of players.

The Adversary. We assume a probabilistic polynomial time *malicious* adversary, who may deviate from the protocol description arbitrarily. The adversary can corrupt up to $t$ players, and it learns the private state of all corrupted players. As in previous threshold ECDSA schemes [4, 17, 18, 28], we limit ourselves to *static* corruptions, meaning the adversary must choose which players to corrupt at the beginning of the protocol. There are standard techniques for converting a protocol secure against static corruptions to secure against adaptive corruptions [6, 25], but these will incur an overhead.

We assume a *rushing* adversary, meaning that the adversary gets to speak last in a given round and, in particular, can choose his message after seeing the honest parties' messages.

Following [4, 17] (but unlike [18]), we assume a **dishonest majority**, meaning $t$, the number of players the adversary corrupts, can be up to $n - 1$. In this case, there is no guarantee that the protocol will complete, and we therefore do not attempt to achieve *robustness*, or the ability to complete the protocol even in the presence of some misbehaving participants.

### 2.1 Signature Schemes

A digital signature scheme $\mathcal{S}$ consists of three efficient algorithms:

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Key\text{-}Gen}(1^\lambda)$, the randomized key generation algorithm which takes as input the security parameter and returns the private signing key $\mathsf{sk}$ and public verification key $\mathsf{pk}$.
- $\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$, the possibly randomized signing algorithm which takes as input the private key $sk$ and the message to be signed $m$ and outputs a signature, $\sigma$. As the signature may be randomized, there may be multiple valid signatures. We denote the set of valid signatures as $\{\mathsf{Sig}(\mathsf{sk}, m)\}$ and require that $\sigma \in \{\mathsf{Sig}(\mathsf{sk}, m)\}$.
- $\mathsf{b} \leftarrow \mathsf{Ver}\ (\mathsf{pk}, m, \sigma)$, the deterministic verification algorithm, which takes as input a public key $\mathsf{pk}$, a message $m$ and a signature $\sigma$ and outputs a bit $b$ which equals 1 if and only if $\sigma$ is a valid signature on $m$ under $\mathsf{pk}$.

To prove a signature scheme secure, we recall the standard notion of existential unforgeability against chosen message attacks (EU-CMA) as introduced in [23].

**Definition 1 (Existential unforgeability).** *Consider a PPT adversary $\mathcal{A}$ who is given public key* $\mathsf{pk}$ *output by* Key-Gen *and oracle access to the signing algorithm* $\mathsf{Sig}(\mathsf{sk}, \cdot)$ *with which it can receive signatures on adaptively chosen messages of its choosing. Let $\mathcal{M}$ be the set of messages queried by $\mathcal{A}$. A digital signature scheme $\mathcal{S}$ =*(Key-Gen,Sig,Ver) *is said to be* existentially unforgeable *if there is no such PPT adversary $\mathcal{A}$ that can produce a signature on a message $m \notin \mathcal{M}$, except with negligible probability in $\lambda$.*

### 2.2 Threshold Signatures

**Threshold secret sharing.** A $(t, n)-$threshold secret sharing of a secret $x$ consists of $n$ shares $x_1, \ldots, x_n$ such that an efficient algorithm exists that takes as input $t + 1$ of these shares and outputs the secret, but $t$ or fewer shares do not reveal any information about the secret.

**Threshold signature schemes.** Consider a signature scheme, $\mathcal{S}$=(Key-Gen, Sig, Ver). A $(t, n)$-threshold signature scheme $\mathcal{TS}$ for $\mathcal{S}$ enables distributing the signing among a group of $n$ players, $P_1, \ldots, P_n$ such that any group of at least $t + 1$ of these players can jointly generate a signature, whereas groups of size $t$ or fewer cannot. More formally, $\mathcal{TS}$ consists of two protocols:

- Thresh-Key-Gen, the distributed key generation protocol, which takes as input the security parameter $1^\lambda$. Each player $P_i$ receives as output the public key $\mathsf{pk}$ as well as a private output $\mathsf{sk}_i$, which is $P_i$'s share of the private key. The values $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ constitute a $(t, n)$ threshold secret sharing of the private key $\mathsf{sk}$.
- Thresh-Sig, the distributed signing protocol which takes as public input a message $m$ to be signed as well as a private input $sk_i$ from each player. It outputs a signature $\sigma \in \{\mathsf{Sig}(\mathsf{sk}, m)\}$.

Notice that the signature output by Thresh-Sig is a valid signature under Sig, the centralized signing protocol. Thus we do not specify a threshold variant of the verification algorithm as we will use the centralized verification algorithm, Ver.

In some applications, it may be acceptable to have a trusted dealer generate the private key shares for each party. In this case, Thresh-Key-Gen would not be run.

Following [18, 19], we present a game-based definition of security analogous to EU-CMA.

**Definition 2 (Unforgeable threshold signature scheme [18]).** *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 $\lambda$) 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 as well as signatures on those messages.*

This is a game-based definition of security which is analogous to the notion of existential unforgeability under chosen message attack as defined by Goldwasser, Micali, and Rivest [23]. Unlike in the centralized EU-CMA definition, the adversary is additionally given the corrupted players' views of the key generation protocol as well as their views in the signing protocol for the messages it chooses. A stronger simulation-based definition is also possible (see e.g. [17, 18, 28]). See Section 6.3 in which we show how to prove security of our protocol using this stronger simulation-based definition.

### 2.3 Additively Homomorphic Encryption

Our protocol relies on an encryption scheme $\mathcal{E}$ that is additively homomorphic modulo a large integer $N$. Let $E_{pk}(\cdot)$ denote the encryption algorithm for $\mathcal{E}$ using public key $\mathsf{pk}$. Given ciphertexts $c_1 = E_{\mathsf{pk}}(a)$ and $c_2 = E_{\mathsf{pk}}(b)$, there is an efficiently computable function $+_E$ such that

$$c_1 +_E c_2 = E_{\mathsf{pk}}(a + b \bmod N)$$

The existence of a ciphertext addition operation also implies a scalar multiplication operation, which we denote by $\times_E$. Given an integer $a \in N$ and a ciphertext $c = E_{\mathsf{pk}}(m)$, then we have

$$a \times_E c = E_{\mathsf{pk}}(am \bmod N)$$

Informally, we say that $\mathcal{E}$ is semantically secure if for the probability distributions of the encryptions of any two messages are computationally indistinguishable.

We instantiate our protocol using the additively homomorphic encryption scheme of Paillier [33], and we recall the details here:

- Key-Gen: 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 $N$, and denote $\Gamma = N + 1$. The public key is $Z_N$ 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 $a \times_E c = c^a \bmod N^2 = E(am \bmod N)$.

The security of Paillier's cryptosystem relies on the $N$-residuosity decisional assumption [33], which informally says that it is infeasible to distinguish random $N$-residues from random group elements in $Z_{N^2}^*$.

### 2.4 Non-Malleable Equivocable 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 to open a commitment in any possible way (we will refer to this also as *equivocate* the commitment). This 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:

- $\mathsf{KG}$ is the key generation algorithm, on input the security parameter it outputs a pair $\{\mathsf{pk}, \mathsf{tk}\}$ where $\mathsf{pk}$ is the public key associated with the commitment scheme, and $\mathsf{tk}$ is called the *trapdoor*.
- $\mathsf{Com}$ is the commitment algorithm. On input $\mathsf{pk}$ 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.
- $\mathsf{Ver}$ is the verification algorithm. On input $C, D$ and $\mathsf{pk}$ it either outputs a message $M$ or $\perp$.
- $\mathsf{Equiv}$ is the algorithm that opens a commitment in any possible way given the trapdoor information. It takes as input $\mathsf{pk}$, 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 $\mathsf{Equiv}$ 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.

Such a commitment is *non-malleable* [13] if no adversary $\mathcal{A}$, given a commitment $C$ to a messages $m$, is able to produce another commitment $C'$ such that after seeing the opening of $C$ to $m$, $\mathcal{A}$ can successfully decommit to a related message $m'$ (this is actually the notion of non-malleability with respect to opening introduced in [10]).

The non-malleable commitment schemes in [10, 11] 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 [8, 15, 31]), and any of them can be used in our threshold DSA scheme.

However in practice one can use any secure hash function $H$ and define the commitment to $x$ as $h = H(x, r)$, for a uniformly chosen $r$ of length $\lambda$ and assume that $H$ behaves as a random oracle. We use this efficient random oracle version in our implementation.

### 2.5 The Digital Signature Standard

The Digital Signature Algorithm (DSA) was proposed by Kravitz in 1991, and adopted by NIST in 1994 as the Digital Signature Standard (DSS) [3, 27]. ECDSA, the elliptic curve variant of DSA, has become quite popular in recent years, especially in cryptocurruencies.

All of our results in this paper apply to both the traditional DSA and ECDSA. We present our results using the generic G-DSA notation from [17], which we recall here.

The Public Parameters consist of a cyclic group $\mathcal{G}$ of prime order $q$, a generator $g$ for $\mathcal{G}$, a hash function $H : \{0,1\}^* \to Z_q$, and another hash function $H' : \mathcal{G} \to Z_q$.

.

Key-Gen On input the security parameter, outputs a private key $x$ chosen uniformly at random in $Z_q$, and a public key $y = g^x$ computed in $\mathcal{G}$.

Sig On input an arbitrary message $M$,

- compute $m = H(M) \in Z_q$
- choose $k \in_R Z_q$
- compute $R = g^{k^{-1}}$ in $\mathcal{G}$ and $r = H'(R) \in Z_q$
- compute $s = k(m + xr) \bmod q$

- output $\sigma = (r, s)$

Ver On input $M, \sigma$ and $y$,

- check that $r, s \in Z_q$
- compute $R' = g^{ms^{-1} \bmod q} y^{rs^{-1} \bmod q}$ in $\mathcal{G}$
- Accept (output 1) iff $H'(R') = r$.

The traditional DSA algorithm is obtained by choosing large primes $p, q$ such that $q|(p-1)$ and setting $\mathcal{G}$ to be the order $q$ subgroup of $Z_p^*$. In this case the multiplication operation in $\mathcal{G}$ is multiplication modulo $p$. The function $H'$ is defined as $H'(R) = R \bmod q$.

The ECDSA scheme is obtained by choosing $\mathcal{G}$ as a group of points on an elliptic curve of cardinality $q$. In this case the multiplication operation in $\mathcal{G}$ is the group operation over the curve. The function $H'$ is defined as $H'(R) = R_x \bmod q$ where $R_x$ is the $x$-coordinate of the point $R$.

### 2.6 Feldman's VSS Protocol

Recall that in Shamir's scheme [36], to share a secret $\sigma \in Z_q$, the dealer generates a random degree $t$ polynomial $p(\cdot)$ over $Z_q$ such that $p(0) = \sigma$. The secret shares are evaluations of the polynomial

$$p(x) = \sigma + a_1 x + a_2 x^2 + \cdots + a_t x^t \bmod q$$

Each player $P_i$ receives a share $\sigma_i = p(i) \bmod q$.

In a verifiable secret sharing scheme, auxiliary information is published that allows players to check that their shares are consistent and define a unique secret.

Feldman's VSS is an extension of Shamir secret sharing in which the dealer also publishes $v_i = g^{a_i}$ in $\mathcal{G}$ for all $i \in [1, t]$ and $v_0 = g^\sigma$ in $\mathcal{G}$.

Using this auxiliary information, each player $P_i$ can check its share $\sigma_i$ for consistency by verifying:

$$g^{\sigma_i} \stackrel{?}{=} \prod_{j=0}^{t} v_j^{i^j} \text{ in } \mathcal{G}$$

If the check does not hold for any player, it raises a complaint and the protocol terminates. Note that this is different than the way Feldman VSS was originally presented as it assumed an honest majority and could recover if a dishonest player raised a complaint. However, since we assume dishonest majority in this paper, the protocol will abort if a complaint is raised.

While Feldman's scheme does leak $g^\sigma$, it can be shown via a simulation argument that nothing else is leaked, but we omit the details here.

### 2.7 Assumptions

DDH. Let $\mathcal{G}$ be a cyclic group of prime order $q$, generated by $g$. The DDH Assumption states that the following two distributions over $\mathcal{G}^3$ are computationally indistinguishable: $DH = \{(g^a, g^b, g^{ab})$ for $a, b \in_R Z_q\}$ and $R = \{(g^a, g^b, g^c)$ for $a, b, c \in_R Z_q\}$.

Strong-RSA. Let $N$ be the product of two safe primes, $N = pq$, with $p = 2p' + 1$ and $q = 2q' + 1$ with $p', q'$ primes. With $\phi(N)$ we denote the Euler function of $N$, i.e. $\phi(N) = (p-1)(q-1) = p'q'$. With $Z_N^*$ we denote the set of integers between 0 and $N-1$ and relatively prime to $N$.

Let $e$ be an integer relatively prime to $\phi(N)$. The RSA Assumption [34] states that it is infeasible to compute $e$-roots in $Z_N^*$. That is, given a random element $s \in_R Z_N^*$ it is hard to find $x$ such that $x^e = s \bmod N$.

The Strong RSA Assumption (introduced in [2]) states that given a random element $s$ in $Z_N^*$ it is hard to find $x, e \neq 1$ such that $x^e = s \bmod N$. The assumption differs from the traditional

RSA assumption in that we allow the adversary to freely choose the exponent $e$ for which she will be able to compute $e$-roots.

We now give formal definitions. Let $SRSA(n)$ be the set of integers $N$, such that $N$ is the product of two $n/2$-bit safe primes.

**Assumption 1** *We say that the Strong RSA Assumption holds, if for all probabilistic polynomial time adversaries $\mathcal{A}$ the following probability*

$$Prob[\ N \leftarrow SRSA(n)\ ;\ s \leftarrow Z_N^*\ :\ \mathcal{A}(N,s) = (x,e)\ \ s.t.\ \ x^e = s \bmod N\ ]$$

*is negligible in $n$.*

## 3 A share conversion protocol

Assume that we have two parties Alice and Bob holding two secrets $a, b \in Z_q$ respectively which we can think of as multiplicative shares of a secret $x = ab \bmod q$. Alice and Bob would like to compute secret additive shares $\alpha, \beta$ of $x$, that is random values such that $\alpha + \beta = x = ab \bmod q$ with Alice holding $\alpha$ (and $a$) and Bob holding $\beta$ (and $b$).

Here we show a protocol based on an additively homomorphic scheme which has appeared many times before in the literature (e.g. [9, 26, 28, 30]) but that we adapt to our needs. We assume that Alice is associated with a public key $E_A$ for an additively homomorphic scheme $\mathcal{E}$ over an integer $N$.

In the following we will refer to this protocol as an MtA (for Multiplicative to Additive) share conversion protocol. In our protocol we also assume that $B = g^b$ might be public. In this case an extra check for Bob is used to force him to use the correct value $b$. We refer to this enhanced protocol as MtAwc (as MtA "with check").

1. Alice initiates the protocol by
   - sending $c_A = E_A(a)$ to Bob
   - proving in ZK that he knows $a < q^3$ via a range proof
2. Bob computes the ciphertext $c_B = b \times_E c_A +_E E_A(\beta') = E_A(ab + \beta')$ where $\beta'$ is chosen uniformly at random in $Z_{q^5}$. Bob sets his share to $\beta = -\beta' \bmod q$. He responds to Alice by
   - sending $c_B$
   - proving in ZK that he knows $b < q^3, \beta' < q^7$ such that $c_B = b \times_E c_A +_E E_A(\beta')$.
   - and *only if $B = g^b$ is public* that $B = g^b$ and
3. Alice decrypts $c_B$ to obtain $\alpha'$ and sets $\alpha = \alpha' \bmod q$.

Correctness. Assume both players are honest and $N > q^8$. Then note that Alice decrypts the value $\alpha' = ab + \beta' \bmod N$. Note that $\beta' < N - ab$ and therefore the reduction mod $N$ is not executed which implies that the protocol correctly computes $\alpha, \beta$ such that $\alpha + \beta = x \bmod q$.

*Simulation.* We first point out that as a stand-alone protocol, we can prove security even without the range proofs. Indeed, if the adversary corrupts Alice, then Bob's message can be simulated without knowledge of its input $b$. Indeed a simulator can just choose a random $b' \in Z_q$ and act as Bob. The distribution of the message decrypted by Alice in this simulation is statistically close to the message decrypted when Bob uses the real $b$, because the "noise" $\beta'$ is uniformly distributed in $Z_{q^5}$.

If the adversary corrupts Bob, then Alice's message can be simulated without knowledge of its input $a$. Indeed a simulator can just choose a random $a' \in Z_q$ and act as Alice. In this case the view of Bob is computationally indistinguishable from the real one due to the semantic security of the encryption scheme $\mathcal{E}$.

However if the range proofs are not used, a malicious Alice or Bob can cause the protocol to "fail" by choosing large inputs. As a stand-alone protocol this is not an issue since the parties are

not even aware that the reduction mod $N$ took place and no information is leaked about the other party's input. However, when used inside our threshold DSA protocol, this attack will cause the signature verification to fail, and this information is linked to the size of the other party's input.

Consider for example the case of Alice running the protocol with input $a' = q^7 + a$. If Bob's input is "small" then the reduction mod $N$ will not take place and the protocol will succeed, and eventually the signature produced by our threshold DSA protocol will verify (since $a' = a \bmod q$). But if Bob's input is large the protocol will fail. Similar issues arise if one does not check the range of $b$ and $\beta'$.

So we need security in the presence of an oracle that tells the parties if the reduction mod $N$ happens or not, but due to the ZK "range proofs" such reduction will only happen with negligible probability and security holds.

REMARK. *On the ZK proofs and the size of the modulus $N$.* For the ZK proofs required in the protocol we use simplified versions of similar ZK proofs presented in [30] (and already used in [17]). These are ZK arguments with security holding under the Strong RSA Assumption. Moreover they require $N \approx q^8$ as pointed above. We point out that for typical choices of parameters, $N$ is approximately $q^8$ (since $q$ is typically 256-bit long while $N$ is a 2048-bit RSA modulus), so this requirement is not problematic. It is however imperative that the size of $N$ is checked by the parties to ensure the ZK property of the proofs[4].

**Note about the previous version:** In our previous version we chose $\beta'$ uniformly at random in $Z_N$ and did not impose any range check on it. This leads to a similar information leakage as described above where if $\beta'$ is chosen close to $N$, a modular reduction (and therefore a failure of the protocol) happens based on the distribution of the input $a$.

## 4 Our scheme

We now describe our protocol. The players run on input $\mathsf{G}, g$ the cyclic group used by the DSA signature scheme. We assume that each player $P_i$ is associated with a public key $E_i$ for an additively homomorphic encryption scheme $\mathcal{E}$.

### 4.1 Key generation protocol

- Phase 1. Each Player $P_i$ selects $u_i \in_R Z_q$; computes $[KGC_i, KGD_i] = \mathsf{Com}(g^{u_i})$ and broadcasts $KGC_i$. Each Player $P_i$ broadcasts $E_i$ the public key for Paillier's cryptosystem.
- Phase 2. Each Player $P_i$ broadcasts $KGD_i$. Let $y_i$ be the value decommitted by $P_i$. The player $P_i$ performs a $(t, n)$ Feldman-VSS of the value $u_i$, with $y_i$ as the "free term in the exponent" The public key is set to $y = \prod_i y_i$. Each player adds the private shares received during the $n$ Feldman VSS protocols. The resulting values $x_i$ are a $(t, n)$ Shamir's secret sharing of the secret key $x = \sum_i u_i$. Note that the values $X_i = g^{x_i}$ are public.
- Phase 3 Let $N_i = p_i q_i$ be the RSA modulus associated with $E_i$. Each player $P_i$ proves in ZK that he knows $x_i$ using Schnorr's protocol [35] and that $N_i$ is square-free using the proof of Gennaro, Micciancio, and Rabin [21].

### 4.2 Signature Generation

We now describe the signature generation protocol, which is run on input $m$ (the hash of the message $M$ being signed) and the output of the key generation protocol described above. We note

[4] For the simple range proof that $a, b < K$ one could alternatively use a variation of Boudot's proof [5] which establish $K \sim q$ which sets $N \sim q^3$. This proof is less efficient that the ones from [17, 30] which are anyway required for Bob in the $\mathsf{MtAwc}$ protocol. Moreover as we said earlier, $N > q^8$ in practice anyway so the improvement in the size of $N$ is irrelevant for ECDSA.

that the latter protocol is a $t$-out-of-$n$ protocol (and thus the secret key $x$ is shared using $(t, n)$ Shamir secret-sharing).

Let $S \subseteq [1..n]$ be the set of players participating in the signature protocol. We assume that $|S| = t + 1$. For the signing protocol we can share any ephemeral secrets using a $(t, t + 1)$ secret sharing scheme, and do not need to use the general $(t, n)$ structure. We note that using the appropriate Lagrangian coefficients $\lambda_{i,S}$ each player in $S$ can locally map its own $(t, n)$ share $x_i$ of $x$ into a $(t, t + 1)$ share of $x$, $w_i = (\lambda_{i,S})(x_i)$, i.e. $x = \sum_{i \in S} w_i$. Since $X_i = g^{x_i}$ and $\lambda_{i,S}$ are public values, all the players can compute $W_i = g^{w_i} = X_i^{\lambda_{i,S}}$.

- Phase 1. Each Player $P_i$ selects $k_i, \gamma_i \in_R Z_q$; computes $[C_i, D_i] = \mathsf{Com}(g^{\gamma_i})$ and broadcast $C_i$. Define $k = \sum_{i \in S} k_i$, $\gamma = \sum_{i \in S} \gamma_i$. Note that

$$k\gamma = \sum_{i,j \in S} k_i \gamma_j \bmod q$$

$$kx = \sum_{i,j \in S} k_i w_j \bmod q$$

- Phase 2. Every pair of players $P_i, P_j$ engages in two multiplicative-to-additive share conversion subprotocols
  - $P_i, P_j$ run MtA with shares $k_i, \gamma_j$ respectively. Let $\alpha_{ij}$ [resp. $\beta_{ij}$] be the share received by player $P_i$ [resp. $P_j$] at the end of this protocol, i.e.

$$k_i \gamma_j = \alpha_{ij} + \beta_{ij}$$

    Player $P_i$ sets $\delta_i = k_i \gamma_i + \sum_{j \neq i} \alpha_{ij} + \sum_{j \neq i} \beta_{ji}$. Note that the $\delta_i$ are a $(t, t + 1)$ additive sharing of $k\gamma = \sum_{i \in S} \delta_i$
  - $P_i, P_j$ run MtAwc with shares $k_i, w_j$ respectively. Let $\mu_{ij}$ [resp. $\nu_{ij}$] be the share received by player $P_i$ [resp. $P_j$] at the end of this protocol, i.e.

$$k_i w_j = \mu_{ij} + \nu_{ij}$$

    Player $P_i$ sets $\sigma_i = k_i w_i + \sum_{j \neq i} \mu_{ij} + \sum_{j \neq i} \nu_{ji}$. Note that the $\sigma_i$ are a $(t, t + 1)$ additive sharing of $kx = \sum_{i \in S} \sigma_i$
- Phase 3. Every player $P_i$ broadcasts $\delta_i$ and the players reconstruct $\delta = \sum_{i \in S} \delta_i = k\gamma$. The players compute $\delta^{-1} \bmod q$.
- Phase 4. Each Player $P_i$ broadcasts $D_i$. Let $\Gamma_i$ be the values decommitted by $P_i$ who proves in ZK that he knows $\gamma_i$ s.t. $\Gamma_i = g^{\gamma_i}$ using Schnorr's protocol [35].
  The players compute

$$R = [\prod_{i \in S} \Gamma_i]^{\delta^{-1}} = g^{(\sum_{i \in S} \gamma_i) k^{-1} \gamma^{-1}} = g^{\gamma k^{-1} \gamma^{-1}} = g^{k^{-1}}$$

  and $r = H'(R)$.
- Phase 5. Each player $P_i$ sets $s_i = mk_i + r\sigma_i$. Note that

$$\sum_{i \in S} s_i = m \sum_{i \in S} k_i + r \sum_{i \in S} \sigma_i = mk + rkx = k(m + xr) = s$$

  i.e. the $s_i$ are a $(t, t + 1)$ sharing of $s$.
  - (5A) Player $P_i$ chooses $\ell_i, \rho_i \in_R Z_q$ computes $V_i = R^{s_i} g^{\ell_i}$, $A_i = g^{\rho_i}$, and $[\hat{C}_i, \hat{D}_i] = \mathsf{Com}(V_i, A_i)$ and broadcasts $\hat{C}_i$.
    Let $\ell = \sum_i \ell_i$ and $\rho = \sum_i \rho_i$.

- (5B) Player $P_i$ broadcasts $\hat{D}_i$ and proves in ZK that he knows $s_i, \ell_i, \rho_i$ such that $V_i = R^{s_i} g^{\ell_i}$ and $A_i^{\rho_i}$. If a ZK proof fails, the protocol aborts. Let $V = g^{-m} y^{-r} \prod_{i \in S} V_i$ (this should be $V = g^{\ell}$) and $A = \prod_{i \in S} A_i$.
- (5C) Player $P_i$ computes $U_i = V^{\rho_i}$ and $T_i = A^{\ell_i}$. It commits $[\tilde{C}_i, \tilde{D}_i] = \mathsf{Com}(U_i, T_i)$ and broadcasts $\tilde{C}_i$.
- (5D) Player $P_i$ broadcasts $\tilde{D}_i$ to decommit to $U_i, T_i$ If $\prod_{i \in S}[T_i] \neq \prod_{i \in S} U_i$ the protocol aborts.
- (5E) Otherwise player $P_i$ broadcasts $s_i$. The players compute $s = \sum_{i \in S} s_i$. If $(r, s)$ is not a valid signature the players abort, otherwise they accept and end the protocol.

Let us explain the intuition behind Phase 5. To avoid expensive ZK proofs, we are potentially reconstructing an incorrect signature, which is then checked and possibly rejected. A naive approach to the last phase is for the players to reveal $s_i$ and reconstruct $s = \sum_i s_i$. But, for reasons that will become clear in the proof, this is not provably secure—the intuitive reason being that if the adversary makes the protocol fail by outputting an invalid signature, then the values $s_i$ held by the good players may give him valuable information.[5] Naively this could be done by first broadcasting $S_i = R^{s_i}$ and check that $\prod_i S_i = R^s = g^m y^r$ according to the DSA verification algorithm. But for similar reasons, this step makes the proof fail. So in our protocol the players mask $R^{s_i}$ with a random value $g^{\ell_i}$. Let $V_i = R^{s_i} g^{\ell_i}$. Then $\prod_i V_i = R^s g^{\ell}$ and therefore $V = g^{\ell}$. The players cannot reveal $g^{\ell_i}$ to check the correctness of $V$ as this would "de-mask" $R^{s_i}$ so we "randomize" the "aggregate" value to $U = g^{\ell\rho}$. Alongside the players compute $g^{\ell\rho}$ via a distributed "Diffie-Hellman" exchange. If this distributed randomized signature verification carries out, then it is safe to release the shares $s_i$, but if the signature does not verify then the protocol aborts here and the values $s_i$ held by the good players are never revealed in the clear.

### 4.3 The Zero-Knowledge Proofs

In step (5B) a player $P$ outputs $V = R^s g^{\ell}$ and $A = g^{\rho}$ and must prove that he knows $s, \ell, \rho$ satisfying the above relationship. The proof for $A$ is the classic Schnorr's proof. For the value $V$ a classic (honest-verifier) ZK proof for this task is as follows:

- The Prover chooses $a, b \in_R Z_q$ and sends $\alpha = R^a g^b$
- The Verifier sends a random challenge $c \in_R Z_q$
- The Prover answers with $t = a + cs \bmod q$ and $u = b + c\ell \bmod q$.
- The Verifier checks that $R^t g^u = \alpha V^c$

### 4.4 Security Proof

In this section we prove the following

**Theorem 1.** *Assuming that*

- *The DSA signature scheme is unforgeable;*
- *The Strong RSA Assumption holds;*
- KG, Com, Ver, Equiv *is a non-malleable equivocable commitment scheme;*
- *the DDH Assumption holds*

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

[5] We do not have an attack but we do not see a way to make a proof work either.

The proof of this theorem will proceed by a traditional simulation argument, in which we show that if there is an adversary $\mathcal{A}$ that forges in the threshold scheme with a significant probability, then we can build a forger $\mathcal{F}$ that forges in the centralized DSA scheme also with a significant probability.

So let's assume that there is an adversary $\mathcal{A}$ that forges in the threshold scheme with probability larger than $\epsilon \geq \lambda^{-c}$.

We assume that the adversary controls players $P_2, \ldots, P_{t+1}$ and that $P_1$ is the honest player. We point out that because we use concurrently non-malleable commitments (where the adversary can see many commitments from the honest players) the proof also holds if the adversary controls less than $t$ players and we have more than 1 honest player. So the above assumption is without loss of generality.

Because we are assuming a rushing adversary, $P_1$ always speaks first at each round. Our simulator will act on behalf of $P_1$ and interact with the adversary controlling $P_2, \ldots, P_n$. Recall how $\mathcal{A}$ works: it first participates in the key generation protocol to generate a public key $y$ for the threshold scheme. Then it requests the group of players to sign several messages $m_1, \ldots, m_\ell$, and the group engages in the signing protocol on those messages. At the end with probability at least $\epsilon$ the adversary outputs a message $m \neq m_i$ and a valid signature $(r, s)$ for it under the DSA key $y$. This probability is taken over the random tape $\tau_{\mathcal{A}}$ of $\mathcal{A}$ and the random tape $\tau_1$ of $P_1$. If we denote with $\mathcal{A}(\tau_{\mathcal{A}})_{P_1(\tau_1)}$ the output of $\mathcal{A}$ at the end of the experiment described above, we can write

$$Prob_{\tau_1, \tau_{\mathcal{A}}}[\, \mathcal{A}(\tau_{\mathcal{A}})_{P_1(\tau_1)} \text{ is a forgery } ] \geq \epsilon$$

We say that an adversary random tape $\tau_{\mathcal{A}}$ is *good* if

$$Prob_{\tau_1}[\, \mathcal{A}(\tau_{\mathcal{A}})_{P_1(\tau_1)} \text{ is a forgery } ] \geq \frac{\epsilon}{2}$$

By a standard application of Markov's inequality we know that if $\tau_{\mathcal{A}}$ is chosen uniformly at random, the probability of choosing a good one is at least $\frac{\epsilon}{2}$.

We now turn to building the adversary $\mathcal{F}$ that forges in the centralized scheme. This forger will use $\mathcal{A}$ as a subroutine in a "simulated" version of the threshold scheme: $\mathcal{F}$ will play the role of $P_1$ while $\mathcal{A}$ will control the other players. $\mathcal{F}$ will choose a random tape $\tau_{\mathcal{A}}$ for $\mathcal{A}$: we know that with probability at least $\frac{\epsilon}{2}$ it will be a good tape. From now on we assume that $\mathcal{A}$ runs on a good random tape.

$\mathcal{F}$ runs on input a public key $y$ for the centralized DSA scheme, which is chosen according to the uniform distribution in $\mathcal{G}$. The first task for $\mathcal{F}$ is to set up an indistinguishable simulation of the key generation protocol to result in the same public key $y$.

Similarly every time $\mathcal{A}$ requests the signature of a message $m_i$, the forger $\mathcal{F}$ will receive the real signature $(r_i, s_i)$ from its signature oracle. It will then simulate, in an indistinguishable fashion, an execution of the threshold signature protocol that on input $m_i$ results in the signature $(r_i, s_i)$.

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 turn to the details of the simulations.

### 4.5 Simulating the key generation protocol

The simulation Sim-Key-Gen is described below. On input a public key $y = g^x$ for DSA the forger $\mathcal{F}$ plays the role of $P_1$ as follows. The forger $\mathcal{F}$ also runs on input a Paillier public key $E$ for which he does not know the matching secret key (this is necessary for when we have to make a reduction to the semantic security of the Paillier encryption scheme).

Simulation: Repeat the following steps (by rewinding $\mathcal{A}$) until $\mathcal{A}$ sends valid messages (i.e. a correct decommitment) for $P_2, \ldots, P_n$ on both iterations.

- $\mathcal{F}$ (as $P_1$) selects a random value $u_1 \in Z_q$, computes $[KGC_1, KGD_1] = \mathsf{Com}(g^{u_1})$ and broadcasts $KGC_1$. $\mathcal{A}$ broadcasts commitments $KCG_i$ for $i > 1$;
- Each player $P_i$ broadcasts $KGD_i$; let $y_i$ be the decommitted value and the accompanying Feldman-VSS ($\mathcal{F}$ will follow the protocol instructions). Each player broadcasts $E_i$. $\mathcal{F}$ broacasts $E_1 = E$.
- Let $y_i$ denote the revealed commitment values of each party. $\mathcal{F}$ *rewinds* the adversary to the decommitment step and
  - changes the opening of $P_1$ to $K\hat{G}D_1$ so that the committed value revealed is now $\hat{y_1} = y \cdot \prod_{i=2}^{n} y_i^{-1}$.
  - simulates the Feldman-VSS with free term $\hat{y_1}$
- The adversary $\mathcal{A}$ broadcasts $K\hat{G}D_i$. Let $\hat{y}_i$ be the committed value revealed by $\mathcal{A}$ at this point (this could be $\perp$ if the adversary refused to decommit).
- The players compute $\hat{y} = \prod_{i=1}^{n} \hat{y}_i$ (set to $\perp$ if any of the $\hat{y}_i$ are set to $\perp$ in the previous step).

We now prove a few lemmas about this simulation.

**Lemma 1.** *The simulation terminates in expected polynomial time and is indistinguishable from the real protocol.*

*Proof (of Lemma 1).* Since $\mathcal{A}$ is running on a good random tape, we know that the probability over the random choices of $\mathcal{F}$, that $\mathcal{A}$ will correctly decommit is at least $\frac{\epsilon}{2} > \frac{1}{2\lambda^c}$. Therefore we will need to repeat the loop only a polynomial number of times in expectation.

The only differences between the real and the simulated views is that $P_1$ runs a simulated Feldman-VSS with free term in the exponent $\hat{y_1}$ for which it does not know the discrete log. But we know (see Section 2.6) that this simulation is identically distributed from the real Feldman-VSS. So the simulation of the protocol is perfect.

**Lemma 2.** *For a polynomially large fraction of inputs $y$, the simulation terminates with output $y$ except with negligible probability.*

*Proof (of Lemma 2).* First we prove that if the simulation terminates on an output which is not $\perp$, then it terminates with output $y$ except with negligible probability. This is a consequence of the non-malleability property of the commitment scheme. Indeed, if $\mathcal{A}$ correctly decommits $KGC_i$ twice it must do so with the same string, no matter what $P_1$ decommits too (except with negligible probability)[6]. Therefore $\hat{y}_i = y_i$ for $i > 1$ and therefore $\hat{y} = y$.

Then we prove that this happens for a polynomially large fractions of input $y$. Let $y_{\mathcal{A}} = \prod_{i=2}^{n} y_i$, i.e.the contribution of the adversary to the output of the protocol. Note that because of non-malleability, this value is determined and known to $\mathcal{F}$ by the time it rewinds the adversary. At that point $\mathcal{F}$ rewinds the adversary and chooses $\hat{y}_1 = y(y_{\mathcal{A}}^{-1})$. Since $y$ is uniformly distributed, we have that $\hat{y}_1$ is also uniformly distributed. Because $\mathcal{A}$ is running on a good random tape we know that at this point there is an $\frac{\epsilon}{2} > \frac{1}{2\lambda^c}$ fraction of $\hat{y}_1$ for which $\mathcal{A}$ will correctly decommit. Since there is a 1-to-1 correspondence between $y$ and $\hat{y}_1$ we can conclude that for a $\frac{\epsilon}{2} > \frac{1}{2\lambda^c}$ fraction of the input $y$ the protocol will successfully terminate.

### 4.6 Signature generation simulation

After the key generation is over, $\mathcal{F}$ must handle the signature queries issued by the adversary $\mathcal{A}$. When $\mathcal{A}$ requests to sign a message $m$, our forger $\mathcal{F}$ will engage in a simulation of the threshold signature protocol. During this simulation $\mathcal{F}$ will have access to a signing oracle that produces DSA signatures under the public key $y$ issued earlier to $\mathcal{F}$.

[6] This property is actually referred to as independence. This is introduced in [20] as a stronger version of non-malleability and then proven equivalent to non-malleability in [4]).

Semi-Correct Executions. Let $k$ be such that $R = g^{k^{-1}}$ and let $\tilde{k}$ be the value defined by the inputs of the players in the MtA and MtAwc protocols. More specifically if $c_i$ is the encryption sent by player $P_i$ in the first round of those protocols, then define $\tilde{k}_i = Dec_i(c_i)$ and $\tilde{k} = \sum_i \tilde{k}_i$.

We say that a protocol execution is semi-correct if in step (4) it holds that $k = \tilde{k}$. Note that this condition is well defined since the values $k, \tilde{k}$ are uniquely determined by step (4). Note that an execution is not semi-correct if the adversary "messes up" the computation of $R$ by revealing wrong shares in the computation of $\delta$.

In the real run of the protocol, it is not feasible to decide if an execution is semi-correct or not. However, as we will see, in the simulation, we can detect whether an execution is semi-correct or not, and depending on whether it is semi-correct the simulator will decide how to simulate Phase 5.

Bird-Eye View of Simulation. First we note that for semi-correct executions the adversary, after Step 4 can already detect if the value $R^{s_1}$ which will be broadcast in Step (5) by the good player is correct or not. In fact by this point the adversary has $s_i$ for $i > 1$ and for a "candidate" $R^{s_1}$ can check if

$$\prod_i R^{s_i} = R^s = g^m y^r$$

Moreover in such executions when we arrive to step (5A) the simulator will be able to "extract" the value $s_1$ for the good player, which will allow the simulation to terminate successfully.

Second, we show that a simulation that is not semi-correct will fail at step (5D) with high probability since the value $U_1$ contributed by the good player is indistinguishable from random. This allows us to simulate Phase (5) by simply using a random $\tilde{s}_1$ for $P_1$.

Finally, the simulator needs to detect whether or not the execution is semi-correct in order to know which path to choose. Although detecting this is not possible in the real protocol, the simulator can extract appropriate values to faciliate this detection. We now proceed with the details.

### 4.7 Semi-correct executions

We now present a simulation that works for a semi-correct execution.

We point out that $\mathcal{F}$ does not know the secret values associated with $P_1$: its correct share $w_1$ of the secret key, and the secret key of its public key $E_1$. The latter is necessary in order to reduce unforgeability to the semantic security of the encryption scheme.

However $\mathcal{F}$ does know the shares $w_j$ of all other players. It also knows the "public key" of $P_1$, $W_1 = g^{w_1}$ from the simulation of the key generation protocol.

In the following simulation $\mathcal{F}$ aborts whenever the protocol is supposed to abort, i.e. if the adversary (i) refuses to decommit in steps 4, 5B or 5D or (ii) fails the ZK proof in Step 2 or 5 or (iii) the signature $(r, s)$ does not verify.

- Phase 1 All the players execute the protocol by broadcasting $C_i$ ($\mathcal{F}$ runs the protocol correctly for $P_1$).
- Phase 2
  - All the players execute the MtA protocol for $k$ and $\gamma$ using the values $k_1$ and $\gamma_1$ that it chose in Phase 1. $\mathcal{F}$ runs the protocol correctly for $P_1$. For each other player $P_{i>1}$, $\mathcal{F}$ runs two MtA protocols, one in which it initiates (using the value $k_1$) and one in which it is the respondent (using the value $\gamma_1$). $\mathcal{F}$ extracts the following values from the range proofs for $i > 1$:
    * $k_i$
    * $\gamma_i$
    * $\beta'_{1i}$

Note that when $\mathcal{F}$ is the initiator, it cannot decrypt its own share $\alpha_{1j}$ during the execution of the protocol with $P_j$ on input $k_1, \gamma_j$. However, using the values it extracted, it can compute $\alpha_{1j} = k_1\gamma_i + \beta'_i \bmod q$.

- All the players execute the MtAwc protocol for $k$ and $x$. Here $\mathcal{F}$ simulates $P_1$ according to the simulation described in Section 3 since it does not know its own value $w_1$.
  Moreover it extracts $P_j$'s resulting share $\nu_{1j}$ from its ZK proof.
- In the protocol with $P_j$ on input $k_j, w_1$, $\mathcal{F}$ does not know $w_1$ so it just sends a random $\mu_{j1}$ to $P_j$.

Note that at this point $\mathcal{F}$ knows the sum of $\sigma_i$ for the bad players. Indeed

$$\sum_{i>1}\sigma_i = \sum_{i,j>1} k_i w_j + \sum_j \mu_{j1} + \sum_j \nu_{1j}$$

and $\mathcal{F}$ knows all the values on the right hand side of the equation.

– Phase 3 All the players execute the protocol by revealing $\delta_i$. Let $\delta = \sum_i \delta_i$ ($\mathcal{F}$ runs the protocol correctly for $P_1$ with the random shares it chose in step 2 – therefore $\mathcal{F}$ is effectively broadcasting a random $\delta_1$).

– Phase 4
  1. Each player reveals $D_i$ to decommit to $\Gamma_i$
  2. $\mathcal{F}$ queries its signature oracle and receives a signature $(r, s)$ on $m$. It computes $R = g^{ms^{-1}} y^{rs^{-1}} \in \mathcal{G}$ (note that $H'(R) = r \in Z_q$).
  3. $\mathcal{F}$ rewinds $\mathcal{A}$ to the decommitment step, and for $P_1$ changes the decommitment to $\hat{\Gamma}_1 = R^{\delta} \prod_{i>1} \Gamma_i^{-1}$. Note that $[\hat{\Gamma}_1 \prod_{i>1} \Gamma_i]^{\delta^{-1}} = R$

  Note that at this point $\mathcal{F}$ knows the value $s_i$ held by the bad players since $s_i = k_i m + \sigma_i r$. So $\mathcal{F}$ can compute the correct $s_1$ held by $P_1$ as $s - \sum_{i>1} s_i$.

– Phase 5 All players execute all the steps in this phase. $\mathcal{F}$ uses $s_1$ as the share for $P_1$.

We prove the following lemma about the simulation.

**Lemma 3.** *Assuming that*

– *The Strong RSA Assumption holds*
– KG, Com, Ver, Equiv *is a non-malleable equivocable commitment;*

*then the simulation has the following properties*

– *on input $m$ it outputs a valid signature $(r, s)$ or aborts.*
– *it is computationally indistinguishable from a semi-correct real execution*

*Proof (of Lemma 3).*

The only differences between the real and the simulated views is the following: In the MtA protocol the values $c_i = E_i(k_i)$ are published and in the real protocol $R = g^{k^{-1}}$ where $k = \sum_i k_i$, while in the simulated execution $R = g^{\hat{k}^{-1}}$ for the $\hat{k}$ chosen by the signature oracle. This is easily seen to be computationally indistinguishable under the semantic security of Paillier's encryption.

Indeed, when $\mathcal{F}$ rewinds the adversary to "fix" the value of $R$, it implicitly changes the value $k_1$ that $\mathcal{F}$ contributes for $P_1$ to $R$. If $R = g^{\hat{k}^{-1}}$, let (implicitly) $\hat{k}_1 = \hat{k} - \sum_{i>1} k_i$. Note that $R^{\hat{k}_1}$ is known since $R^{\hat{k}_1 + \sum_{i>1} k_i} = g$, therefore $R^{\hat{k}_1} = gR^{-\sum_{i>1} k_i}$. So to distinguish between the real execution and the simulated one, the adversary should detect if the ciphertext sent by $\mathcal{F}$ for $P_1$ in the first round of the MtAwc protocol contains a random $k_1$ or the random $\hat{k}_1$ determined as $\log_R(gR^{-\sum_{i>1} k_i})$ which is infeasible under the semantic security of Paillier's encryption (given that all values are proven to be "small" and no wraparound mod $N$ happens).

Note that we are simulating a semi-correct execution with an execution which is *not* semi-correct, but that's okay because the two are indistinguishable.

However, because the real execution is a semi-correct one, we know that the correct shares of $k$ for the adversary are the $k_i$ that the simulator knows. Therefore the value $s_1$ computed by the simulator is consistent with a correct share for $P_1$ for a valid signature $(r, s)$, which makes Phase 5 indistinguishable from the real execution to the adversary.

Let $(r, s)$ be the signature that $\mathcal{F}$ receives by its signature oracle in Step 2 of Phase 4. This is a valid signature for $m$. We prove that if the protocol terminates, it does so with output $(r, s)$. This is a consequence of the non-malleability property of the commitment scheme. Indeed, if the adversary correctly decommits, its openings must be the same except with negligible probability.

### 4.8 Simulation of a non semi-correct execution

We now show how to simulate the last execution for a non semi-correct execution when $\tilde{k} \neq k$. Details follow.

- Phases 1 to 3 The simulator runs the semi-correct simulation through Phase 3 (including aborting at Phase 4 if the adversary fails to decommit).
- Phase 4 $\mathcal{F}$ does not rewind the adversary to "fix" the value of $R$, but runs the protocol normally for $P_1$.
- Phase 5 $\mathcal{F}$ chooses $\tilde{s}_1 \in_R Z_q$ and runs Phase 5 with this value instead of $s_1$, and choosing $U_1$ as a random group element.

Before we prove that this simulation is indistinguishable for non-semi-correct executions let us give an intuition. Note that the only difference with the previous simulation is that here $\mathcal{F}$ uses a random share $\tilde{s}_1$ instead of the $s_1$ that it computed in the other simulation. The reason is that the value $s_1$ computed in the previous simulation is only guaranteed to be the "correct" share of $s$ if the execution is semi-correct. If the adversary shares $k_i$ don't match anymore the value $R$ then $s_1$ is incorrect, and therefore $\mathcal{F}$ chooses a random value instead. In turn this causes $U_1$ to be uniformly distributed and the check in step (5D) to fail.

The main point of the proof is that if the execution is not semi-correct then the value $U_1$ is (given the view of the adversary) computationally indistinguishable from uniform even in the real execution (under the DDH assumption).

Our proof reflects the above intuition. First we prove that a real non-semi-correct execution is indistinguishable from one in which $P_1$ outputs a random $U_1$. And then we prove that this is indistinguishable from the simulation above, where the good player uses a random $\tilde{s}_1$ instead of the correct $s_1$.

**Lemma 4.** *Assuming that*

- KG, Com, Ver, Equiv *is a non-malleable equivocable commitment;*
- *the DDH Assumptions holds*

*then the simulation is computationally indistinguishable from a non-semi-correct real execution*

*Proof (of Lemma 4).*

We construct three games between the simulator (running $P_1$) and the adversary (running all the other players). In $G_0$ the simulator will just run the real protocol. In $G_1$ the simulator will follow the real protocol but will choose $U_1$ as a random group element. In $G_2$ the simulator will run the above simulation.

Indistinguishability of $G_0$ and $G_1$. Let us assume that there is an adversary $\mathcal{A}_0$ that can distinguish between $G_0$ and $G_1$. We show how this contradicts the DDH Assumption.

Let $\tilde{A} = g^a, \tilde{B} = g^b, \tilde{C} = g^c$ be the DDH challenge where either $c = ab$ or is chosen at random in $Z_q$.

The distinguisher $\mathcal{F}_0$ runs $\mathcal{A}_0$, simulating the key generation phase so that $y = \tilde{B} = g^b$. It does that by rewinding the adversary at the end of Phase 2 of the key generation protocol and changing the decommitment of $P_1$ to $y_1 = b\prod_{i>1} y_i^{-1}$.

$\mathcal{F}_0$ also extracts the values $x_i$ from the adversary via the proof of knowledge at the end of the key generation. Note that at this point $y = \tilde{B}$ and $\mathcal{F}_0$ knows $x_i$, but not $b$ and therefore not $x_1$. In this simulation $\mathcal{F}_0$ does know the secret key matching $E_1$ (since we are not making any reduction to the security of the encryption scheme).

Then $\mathcal{F}_0$ runs the signature generation protocol for a not-semi-correct execution. Remember here we assume that we have a $(t, t+1)$ sharing of the secret key. So $b = \sum_{i\in S} w_i$ with $\mathcal{F}_0$ knowing $w_i$ for $i > 1$ but not knowing $w_1$. Denote with $w_A = \sum_{i>1} w_i$ (which is known to $\mathcal{F}_0$) and therefore $w_1 = b - w_A$.

$\mathcal{F}_0$ runs the protocol normally for Phases 1,2,3, and 4. It extracts the value $\gamma_i$ for $i > 1$ (and he knows $\gamma_1$ since he ran $P_1$ normally). Therefore $\mathcal{F}_0$ knows $k$ such that $R = g^{k^{-1}}$ since $k = (\sum_i \gamma_i)\delta^{-1}$. It also knows $k_1$ since it was chosen normally according to the protocol. Before moving to the simulation of Phase 5, let's look at the MtAwc protocol for the computation of the shares $\sigma_i$.

We note that since $\mathcal{F}_0$ knows the decryption key for $E_1$ he also knows all the shares $\mu_{1j}$ from the invocation of the MtAwc protocol between $P_1$ and $P_j$ on input $k_1$ and $w_j$ respectively[7].

For the MtAwc protocol between $P_1$ and $P_j$ on input $w_1$ and $k_j$ respectively, $\mathcal{F}_0$ knows the value $k_j$ input by $P_j$ since he extracts it from the range proof in the MtA protocol, which is also a proof of knowledge of $k_j$. However $\mathcal{F}_0$ does not know $w_1$, so he therefore sends a random $\mu_{j1}$ to $P_j$ and sets (implicitly) $\nu_{j1} = k_j w_1 - \mu j1$.

At the end we have that the share $\sigma_1$ held by $P_1$ is

$$\sigma_1 = k_1 w_1 + \sum_{j>1} \mu_{1j} + \sum_{j>1} \nu_{j1}$$

by rearranging the terms and substituting the above we get

$$\sigma_1 = \tilde{k} w_1 + \sum_{j>1} \mu_{1j} - \sum_{j>1} \mu_{j1}$$

where $\tilde{k} = \sum_i k_i$. Remember that since this is not a semi-correct execution then $\tilde{k} \neq k$ where $R = g^{k^{-1}}$.

Since $w_1 = b - w_A$ we have

$$\sigma_1 = \tilde{k} b + \mu_1$$

where

$$\mu_1 = \sum_{j>1} \mu_{1j} - \sum_{j>1} \mu_{j1} - \tilde{k} w_A$$

with $\mu_1, \tilde{k}$ known to $\mathcal{F}_0$.

Note that this allows $\mathcal{F}_0$ to compute the correct value

$$g^{\sigma_1} = \tilde{B}^{\tilde{k}} g^{\mu_1}$$

and therefore the correct value of $R^{s_1}$ as

$$R^{s_1} = R^{k_1 m + r\sigma_1} = g^{k^{-1}(k_1 m + r\sigma_1)} = g^{k^{-1}(k_1 m + r\mu_1)} \tilde{B}^{k^{-1}\tilde{k}r}$$

or

$$R^{s_1} = g^{\hat{\mu}_1} \tilde{B}^{\hat{\beta}_1}$$

where $\hat{\mu}_1 = k^{-1}(k_1 m + r\mu_1)$ and $\hat{\beta}_1 = k^{-1}\tilde{k}r$ and $\hat{\mu}_1$ and $\hat{\beta}_1$ are known to $\mathcal{F}_0$.

We now continue the simulation

[7] In this case we do not need to extract anything from $P_j$'s ZK proof, but we still need to check that the value sent by $P_j$ is correct.

- 5A/5B $\mathcal{F}_0$ selects a random $\ell_1$ and sets $V_1 = R^{s_1} g^{\ell_1}$ $A_1 = g^{\rho_1} = \tilde{A} = g^a$. It simulates the ZK proof (since it does not know $\rho_1$ or $s_1$). It extracts $s_i, \ell_i, \rho_i$ from the adversary such that $V_i = R^{s_i} g^{\ell_i} = g^{k^{-1} s_i} g^{\ell_i}$ and $A_i = g^{\rho_i}$. Let $s_A = \sum_{i>1} k^{-1} s_i$
  Note that
  $$V = g^{-m} y^{-r} \prod_i V_i = g^{-m} y^{-r} V_1 \prod_{i>1} V_i$$
  and therefore substituting the above relations (and setting $\ell = \sum_i \ell_i$ )
  $$V = g^{\ell} R^{s_1} g^{s_A - m} y^{-r}$$
  Note that $y = \tilde{B}$ so $y^{-r} = \tilde{B}^{-r}$. Therefore
  $$V = g^{\ell} g^{\hat{\mu}_1} \tilde{B}^{\hat{\beta}_1} g^{s_A - m} \tilde{B}^{-r}$$
  or
  $$V = g^{\ell} g^{\theta} \tilde{B}^{\kappa}$$
  where $\theta = \hat{\mu}_1 + s_A - m$ and $\kappa = \hat{\beta}_1 - r$ known to $\mathcal{F}_0$.
  Note that for executions that are not semi-correct $\kappa \neq 0$
- 5C/5D $\mathcal{F}_0$ computes $T_1 = A^{\ell_1}$ correctly (which he can do since he knows $\ell_1$) but for $U_1$ outputs $U_1 = \tilde{A}^{\ell+\theta} \tilde{C}^{\kappa}$ and it aborts.

Note what happens when $\tilde{C} = g^{ab}$. By our choice of $a = \rho_1$ and $b = x$ we have that $U_1 = V^{\rho_1}$ as in Game $G_0$. However when $\tilde{C}$ is a random group element, $U_1$ is uniformly distributed as in $G_1$.

Therefore under the DDH assumption $G_0$ and $G_1$ are indistinguishable.

Indistinguishability of $G_1$ and $G_2$. We note that in $G_2$ the simulator broadcasts a random $\tilde{V}_1 = R^{\tilde{s}_1} g^{\ell_1}$ which is indistinguishable from the correct $V_1 = R^{s_1} g^{\ell_1}$ because of the "mask" $g^{\ell_1}$ which (under the DDH) is computationally indistinguishable from a random value, given that the adversary only has $A_1$.

More in detail, let $\tilde{A} = g^{a-\delta}$, $\tilde{B} = g^b$ and $\tilde{C} = g^{ab}$ be the DDH challenge where $\delta = 0$ or random in $Z_q$.

The simulator here proceeds as in $G_0$ (i.e. the regular protocol) until Phase 5.

- 5A/5B $\mathcal{F}_0$ broadcasts $V_1 = R^{s_1} \tilde{A}$ and $A_1 = \tilde{B}$. It simulates the ZK proof (since it does not know $\ell_1$ or $\rho_1$). It extracts $s_i, \ell_i, \rho_i$ from the adversary such that $V_i = R^{s_i} g^{\ell_i} = g^{k^{-1} s_i} g^{\ell_i}$ and $A_i = g^{\rho_i}$.
- 5C/5D $\mathcal{F}_0$ computes $U_1$ as a random element and $T_1 = \tilde{C} \tilde{A}^{\sum_{j>1} \rho_j}$ and it aborts.

Note what happens when $\tilde{A} = g^a$. By our choice, $a = \ell_1$ and $b = \rho_1$, and we have that $V_1 = R^{s_1} g^{\ell_1}$ and $T_1 = A^{\ell_1}$ as in Game $G_1$. However when $\tilde{A} = g^a g^{-\delta}$ with a random $\delta$, then this is equivalent to have $V_1 = R^{\tilde{s}_1} g^{\ell_1}$ and $T_1 = A^{\ell_1}$ with a randomly distributed $\tilde{s}_1$ as in Game $G_2$.

Therefore under the DDH assumption $G_1$ and $G_2$ are indistinguishable.

### 4.9 Finishing up the proof

Before we conclude the proof we note that our protocol detects the presence of a malicious adversary by noticing that the signature does not verify. As pointed out by Lindell in [28] this strategy is not immediately simulatable against a malicious adversary for the following reason. Consider what happens in Phase 5: In the semi-correct simulation $\mathcal{F}$ rewinds the adversary to "hit" the correct $s$. But if the adversary had decided to be malicious and terminate the protocol with an invalid signature, then the protocol would not be simulatable. If $\mathcal{F}$ hits an invalid signature "on purpose" (e.g. by not rewinding), then the simulation is distinguishable by a semi-honest adversary who does hit the correct signature.

Luckily for a "game-based" definition of security, this is not an issue as discussed in [28]. Let $Q < \lambda^c$ be the maximum number of signature queries that the adversary makes. In the real protocol, the adversary will output a forgery after $\ell < Q$ queries, either because it stops submitting queries, or because the protocol aborts. Therefore in our simulation, following Lindell [28], we choose a random index $\iota \in [0...Q]$:

- if $\iota = 0$ we assume that all executions are semi-correct. In this case we can always simulates as in the previous section
- otherwise we assume that the first $\iota - 1$ executions are semi-correct, but at the $\iota^{th}$ execution the value $V$ is not equal to $g^\ell$.

With probability $1/(Q+1) \geq \lambda^{-c}$ this is a correct guess.

We can now complete the proof.

*Proof (of Theorem 1).*

Unforgeability. The forger $\mathcal{F}$ described above produces an indistinguishable view for the adversary $\mathcal{A}$, and therefore, $\mathcal{A}$ will produce a forgery with the same probability as in real life. The success probability of $\mathcal{F}$ is at least $\frac{\epsilon^3}{8Q}$ where $Q$ is the maximum number of queries. That's because $\mathcal{F}$ has to succeed in

- choosing a good random tape for $\mathcal{A}$ (this happens with probability larger than $\frac{\epsilon}{2}$)
- hitting a good public key $y$ (this also happens with probability larger than $\frac{\epsilon}{2}$)
- guessing the correct index query $\ell$ (this happens with probability larger than $1/Q$)

Under those conditions, the adversary $\mathcal{A}$ will output a forgery with probability at least $\frac{\epsilon}{2}$.

Under the security of the DSA signature scheme, the probability of success of $\mathcal{F}$ must be negligible, which implies that $\epsilon$ must also be negligible, contradicting the assumption that $\mathcal{A}$ has a non-negligible probability of forging.

Correctness. If all players are honest, the protocol fails only if one of the MtA protocols fails. Since we have a total of $4n^2$ such sub-protocols executed during a run of our signature protocol, we have that our protocol fails with probability at most $\frac{4n^2}{q}$ which is negligible.

## 5 Removing the ZK proofs from the MtA protocol

**Note about the previous version:** The protocol described in this section is not secure as shown in [37, 32]. We are leaving the section in the paper as historical record.

As we mentioned in the Introduction, the ZK proofs in the MtA protocol are the most expensive step of our protocol due not only to the fact that these are ZK proofs over the Paillier cryptosystem, but also that every player has to run $n$ of them (since they are specific to each execution of the MtA protocol).

We consider what happens if the range proofs are eliminated. As we discussed in Section 3, the MtA protocol needs to be secure in the presence of an oracle that tells the parties if a reduction mod $N$ happens during the execution. Note that in reality the oracle represents the failure of the verification of the signature generated by the protocol, and if that happens the system is reset. So the oracle is a very weak oracle, which stops working the moment it tells you that a reduction mod $N$ happened.

We conjecture that our protocol remains secure even if the ZK proofs are eliminated for Alice and simplified for Bob in the MtA and MtAwc protocol. More precisely both the MtA and MtAwc protocol work as follows:

- Neither party proves that their values $a, b$ are "small"

– Bob broadcasts $B = g^b, B' = g^{\beta'}$ together with a ZK proof of knowledge for $b, \beta' \bmod q$ using Schnorr's prooof [35]. Alice also checks that $g^\alpha = B^a B'$.
We point out that $B = g^b$ is public in our threshold DSA protocol. Indeed in one case $b = w_i$, the share of the secret key $x$ held by player $P_i$ and $B = g^b$ is public at the end of the key generation phase together with a ZK proof of knowledge. In the other case $b = \gamma_i$, and $B = g^b$ will be public at the end of following round which is when Alice performs the above check.

For sake of completeness here is a description of the sMtA (simplified MtA) protocol which will replace the MtA and MtAwc protocols in our full protocol.

1. Alice initiates the protocol by
   – sending $c_A = E_A(a)$ to Bob
2. Bob computes the ciphertext $c_B = b \times_E c_A +_E E_A(\beta') = E_A(ab + \beta')$ where $\beta'$ is chosen uniformly at random in $Z_N$. Bob sets his share to $\beta = -\beta' \bmod q$. He responds to Alice by
   – sending $c_B$ and $B' = g^{\beta'}$
   – proving in ZK that he knows $b, \beta'$ such that $B = g^b$ and $B' = g^{\beta'}$
3. Alice decrypts $c_B$ to obtain $\alpha'$. She sets $\alpha = \alpha' \bmod q$ and accepts only if $g^\alpha = B^a B'$.

To support our conjecture we propose some "ad-hoc" computational assumptions, which if true would guarantee the security of the protocol. The assumptions are new and non-standard, yet they look reasonable. We discuss them informally below – a full proof of security will appear in the final version.

Information Leaked to Alice by removing the Range Proof. If we remove the proofs that the input $a$ used by Alice is small, we leak information about the input used by Bob via the knowledge of whether a reduction mod $N$ happened or not (see discussion in Section 3).

The standard approach to model this leakage is to give the simulator access to an oracle that tells if the reduction happened or not. Recall that the Bob simulator does not know the inout $b$ and instead chooses a random input $b'$ and a random mask $\beta'$. However the simulator also queries a modular reduction oracle with values $N, a, \beta'$ and the oracle will tell the simulator if $ab + \beta'$ is less than $N$. The output of the simulator includes this "flag" bit.

The oracle will stop working once a moduar reduction takes place (since in the main protocol there will be an abort).

Information Leaked to Bob by removing the ZK Consistency Proof. Here instead we are able to simulate Bob's view under a stronger assumption on the Paillier cryptosystem.

If Bob is corrupted, then the simulated Alice sends the encryption of a random value $c_A = E(\hat{a})$. But then it must decide whether to accept or reject at the end of step (2) (where the real Alice checks that $g^\alpha = B^a B'$) *without knowing* $\hat{a}$. Here we assume that the simulator is provided with an oracle $\Omega_{c_A}(c_B, b, \beta)$ which answers 1 if and only if $Dec(c_B) = b \cdot Dec(c_A) + \beta \bmod q$. Then the simulator will extract $b, \beta$ from the malicious Bob's proof of knowledge, and query $\Omega_{c_A}(c_B, b, \beta)$, and it accepts if the oracle answers 1.

Security cannot be based on the semantic security of the Paillier encryption scheme anymore since the presence of the oracle immediately implies that Paillier is not semantically secure anymore. However consider the following experiment:

– Generate a Paillier key $(E, D)$
– Generate two random values $a_0, a_1 \in_R Z_q$ and publish $A = g^{a_0}$
– Choose a random bit $b$ and publish $c = E(a_b)$
– Let $b'$ be the output of the adversary who is allowed *restricted* access to the oracle $\Omega_c$ – by restricted we mean that the oracle will stop working after it outputs 0.

We say that the Paillier-ECR assumption holds if for every PPT adversary, the probability that $b = b'$ is negligibly close to 1/2. Under the Paillier-ECR assumption we can prove that no adversary given $g^{a_0}$ can distinguish if the sMtA protocol was run with $a_0$ or $a_1$ (with both values being "high

entropy"—in particular randomly chosen). This is sufficient to simulate sMtA with high entropy inputs, which is what is needed to prove security of our threshold DSA protocol.

We note that our Paillier-ECR assumption is a weaker version of the Paillier-EC assumption in [28]. In the latter the oracle access is not restricted, which makes the assumption much stronger. In our case it is sufficient to consider the restricted oracle since the real protocol stops if Alice detects cheating.

### 5.1 Modified simulation for the threshold protocol

We first point out that in the presence of the modular reduction oracle and under the Paillier-ECR assumption, the sMtA protocol is simulatable so the simulation of the threshold DSA protocol remains basically the same, with one crucial difference. When in any of the sMtA simulation the modular reduction oracle flags that a reduction $\bmod N$ took place, the main simulator immediately switches to a non-semi-correct execution since in this case the shares of $k$ held by the players do not match the ones used in the signature computation (i.e. the signature verification is going to fail). Because in real life the protocol aborts the simulation will also stop at this point.

**Remark:** Note that the simulation of the main protocol uses the range proofs also to extract values known by the adversary. Since we removed the range proofs, obviously our simulator cannot do that. However we can augment the key generation protocol with a proof of knowledge of the secret key of the Paillier encryption key. This will allow the simulator to extract the secret keys held by the adversary and it is easy to verify that this will enable the simulator to decrypt the values that in the main simulation were extracted from the range proofs.

### 5.2 Security of the simplified protocol

The two new assumptions that we introduce are very different in nature. The Paillier-ECR assumption just makes a stronger requirement on the Paillier encryption scheme. It is orthogonal to the security of the DSA signature scheme.

On the other hand, assuming the presence of the modular reduction oracle implies stronger security assumptions on the unforgeability of DSA. Indeed note that when the honest player plays the respondent (Bob) role in the sMtA protocols, his inputs are his share of $\rho$ (the mask for the inversion of $k$) and his share of $x$ (the secret key). That means that the protocol is leaking information about these values, and that we need to assume that DSA remains unforgeable even when this information is leaked to the adversary.

To make things more complicated, the adversary controls $n-1$ players, each with its own Paillier modulus $N_i$, and therefore gets information about $\rho$ and $x$ from each sMtA interaction the adversary has with the honest player (each over a different modulus $N_i$).

On the positive side, the shares of $\rho$ and $x$ are "high entropy" secrets and a reduction mod $N$ can only happen once in any of the invocations of the sMtA protocol, since if that happens the protocol ends. It is therefore plausible to assume that we do not leak enough information to allow the adversary to forge.

This can be formalized via the following stronger assumption on the unforgeability of DSA. We define a game between a Challenger and an Attacker:

- The Challenger receives a random DSA public key $y = g^x$ and gives the attacker a random number $\hat{x} \in_R Z_q$. Let $x_1 = x - \hat{x} \bmod q$.
- The Attacker chooses $n-1$ RSA moduli $N_i > q^3$ for $i = 2, \ldots, n$.
- The Attacker submits a message $m$ and $2(n-1)+1$ numbers $\lambda_{i1}, \lambda_{i2} \in Z_N$ and $\hat{\rho} \in Z_q$ for $i = 2, \ldots, n$.
- The Challenger chooses $\rho_1 \in_R Z_q$ and $\beta_{i1}, \beta_{i2} \in_R Z_N$ for $i = 2, \ldots, n$..

- The Challenger has access to an oracle that if all the values $\lambda_{i1}x_1 + \beta_1$ and $\lambda_{i2}\rho_1 + \beta_2$ are less than $N$, will return $(r, s)$ a valid DSA signature on $m$ and also $\alpha = \rho k \bmod q$ where $k \in_R Z_q$ and $r = g^{k^{-1}}$ and $\rho = \hat{\rho} + \rho'$. In this case these values are returned to the Challenger.
- Otherwise the oracle returns nothing and the game ends.

The Attacker wins if he forges a signature on a message for which the Challenger did not output a signature. We say that the Modular DSA Security Assumption holds if the probability that an efficient adversary wins the above game is infeasible.

**Theorem 2.** *Assuming*

- *The Modular DSA Security Assumption;*
- *The Paillier-ECR Assumption;*
- *that* KG, Com, Ver, Equiv *is a non-malleable equivocable commitment scheme;*
- *the DDH Assumption*

*then our threshold DSA scheme where the* sMtA *protocol replaces the* MtA *and* MtAwc *protocols, is unforgeable.*

To prove the theorem we show that an adversary who is able to forge a DSA signature in our simplified protocol, can be turned into an attacker breaking the Modular DSA Security Assumption.

Let $y = g^x$ the DSA public key received by the Challenger. The Challenger runs a simulation for the distributed Adversary which ends with $y$ as the public key. This simulation is identical to the one described in Section 4.5. Here too we assume that the adversary controls players $2, \ldots, n$ and that the simulator controls player 1. The Adversary now knows $\hat{x}$ the sum of the shares of the bad players but not $x_1$ the share of the good player with $x = x_1 + \hat{x} \bmod q$.

When the Adversary submits a message $m$, the Challenger runs a simulation similar to the one described in Section 4.6.

Here the value $\lambda_{i1}$ is the value $a$ submitted by player $P_i$ (controlled by the adversary) as the initiator in the first instance of the sMtA protocol with $P_1$ running on input $x_1 - \beta_{i1}$ is the mask $\beta'$ used by $P_1$ as the respondent in that same protocol.

Similarly the value $\lambda_{i2}$ is the value $a$ submitted by player $P_i$ (controlled by the adversary) as the initiator in the second instance of the sMtA protocol with $P_1$ running on input $\rho_1 - \beta_{i2}$ is the mask $\beta'$ used by $P_1$ as the respondent in that same protocol.

Now the oracle allowed to the Challenger is exactly the Modular Reduction Oracle needed to simulate $P_1$ in the sMtA protocol. Therefore the Challenger can complete the simulation, with the only difference that if the oracle signals that a modular reduction happened in any of the protocols, the simulation follows the non-semi-correct case.

This simulation is clearly indistinguishable from the real execution, therefore the Adversary will forge, and therefore we have created a succesful attacker for the Modular DSA game.

## 6 Extensions

Here we present the following natural extensions to our result.

### 6.1 Other additively homomorphic schemes.

Our scheme works with any additively homomorphic scheme with no modification. It requires an assumption analogous to the Paillier-EC or an efficient ZK Proof for the statement in the MtAwc protocol.

We also note that it is important that security holds under "adversarially chosen" public keys (i.e. we need to prove or assume that the adversary cannot generate a public key such that it gives him and advantage in the MtA protocol).

### 6.2 Other multiplicative to share conversions.

Our threshold DSA scheme works with any MtA protocol, i.e. any protocol that allows two parties to convert their multiplicative shares of a secret into additive shares.

In particular the classic approach based on oblivious transfer by Gilboa [22] can be used. The original protocol in [22] is secure only against semi-honest adversaries, but it can be strengthened against a malicious adversary (see the literature on SPDZ or the recent work on threshold DSA in [12]).

### 6.3 Simulation-Based Security

Our proof uses the game-based definition of unforgeability. The main technical reason is that the simulator cannot detect if the current execution is semi-correct or not, and therefore has to guess. This prevents us from achieving the stronger notion of simulation-based security (where each execution of the protocol can be fully simulated).

While in the real world it is unfeasible to decide if an execution is semi-correct or not, the simulator can do that if it were able to "extract" the bad players' inputs to the MtA protocols. Indeed that would allow the simulator to check that the values $\delta_i, \Gamma_i$ sent by the bad players in Phases 3 and 4 are consistent with the inputs entered in the MtA protocols. If they are, the execution is semi-correct, if they are not then the execution is not semi-correct. Once the simulator knows which execution it is, it can choose the correct simulation strategy.

We note that in our current simulation, the simulator can already extract the input $k_i$ (from the range proof in the MtA protocol) and the input $w_i$ (the share of the secret key, which it extracted during the key generation). But in our current simulation it is not able to extract $\gamma_i$ since we do not require the players to prove knowledge of it. This is the section we discussed. Even in the current protocol, we can extract $\gamma_i$ from the range proof and indeed we do this in our simulation. So is our protocol simulatable as is? And if it is should we just modify the main proof to no longer guess whether the execution is semi-correct but to use this technique

The best way to solve this is to require $P_i, P_j$ to run MtAwc also when interacting on inputs $k_i, \gamma_j$, since MtAwc forces the respondent (which runs on input $\gamma_i$) to prove knowledge of its input. In turn this will allow the simulator to extract $\gamma_i$ for the bad players and detect what kind of execution is being run.

We note that the maliciously secure OT-based MtA protocol from [12] also allows for input extraction, and therefore if used in our protocol, it will yield a fully simulatable protocol.

### 6.4 Deterministic Key Generation

A very popular feature of Bitcoin wallets is deterministic key generation. Introduced in Bitcoin-Improvement-Proposal 32 (BIP32), the idea of this scheme is to allow one to deterministicly generate many keys from a single ECDSA key. Our key sharing is compatible with BIP32 public derivations, and we leave it as future work to prove security in this setting.

## 7 Implementation, Benchmarks, and Evaluation

We implemented both the key generation and signature generation of our protocol, and we confirm that they are highly efficient and fast enough to be used in practice. We benchmarked the version of our protocol from Section 5 that does not contain the range proofs, but relies on the Paillier-ECR assumption. We compare the performance of our protocol to the runtimes of Gennaro *et al.* [17] and Boneh *et al.* [4]. All benchmarks were single-threaded and run on an an Intel quad-core i7-6700 CPU @ 3.40GHz and 64GB of RAM. We ran the code [17] and [4] on our benchmark machine to get an accurate comparison. It should be noted that we implemented our scheme in C while theirs

is a Java implementation which calls native C libraries for the heaviest arithmetic computations. All benchmarks were taken over the secp256k1 curve, which is is the curve used in Bitcoin and more recently a NIST standard.

For the curve operations, we used libsecp256k1.[8] We implemented the MtA protocol with Paillier using the implementation from libhcs.[9].

### 7.1 Benchmarking the data complexity

When compared to [17, 4], we reduce the amount of data transmitted. All figures in this section were measured empirically from the respective implementations, and thus it is possible that they may be further optimized in practice. For a threshold of $t$ (i.e. when there are $t + 1$ participants in the signing protocol), the total data $d$ in bytes sent and received by a given player to/from all other players during the signing protocol is given by: '

$$d_{ours}(t) = 2,328 + t \times 5,024 \textbf{ Bytes}$$

In contrast, the data sent to/from a given player in [17] is given by:

$$d_{Gennaro}(t) = (t + 1) \times 34,578 \textbf{ Bytes}$$

And the data transmitted per player in [4] is given by:

$$d_{Boneh}(t) = (t + 1) \times 38,189 \textbf{ Bytes}$$

Lastly, we mention that for the 2-of-$n$ case, we have $d_{ours}(t = 1) = 3,976$ **B**. In contrast, the recent protocol of [12] requires far more than that with 86.7 **KiB** for 2-of-2 signing and 106.7 **KiB** for 2-of-$n$ signing. Lindell's scheme [28] only requires 769 **B** to be communicated in the 2-of-2 case (but does not support 2-of-$n$).

### 7.2 Benchmarking signature generation time

Following the methodology of [4, 17], we benchmark the raw computation time of a single player without counting network costs. Since each player runs their computation in parallel, this represents the running time of the entire protocol other than network latency. We find that our protocol significantly outperforms both of [4, 17] when using this metric.

As in [4, 17], the protocol running time has a fixed cost that is independent of the number of players plus a linear marginal cost as the threshold increases. We stress that the signing time only depends on the number of active participants $(t + 1)$, but does not depend on $n$, the total number of players. All times are given on a single core, and were averaged over 1000 iterations.

Our protocols running time is given by:

$$r_{ours}(t) = 29 + (t) \times 24 \textbf{ milliseconds}$$

The running time of [17] is given by:

$$r_{Gennaro}(t) = 142 + (t) \times 52 \textbf{ milliseconds}$$

The running time of [4] is given by:

$$r_{Boneh}(t) = 397 + (t) \times 91 \textbf{ milliseconds}$$

We can see that our protocol significantly outperforms both previous schemes. See Figure 1 for a comparison of the concrete raw computation times for thresholds up to 20.

[8] https://github.com/bitcoin-core/secp256k1
[9] https://github.com/tiehuis/libhcs

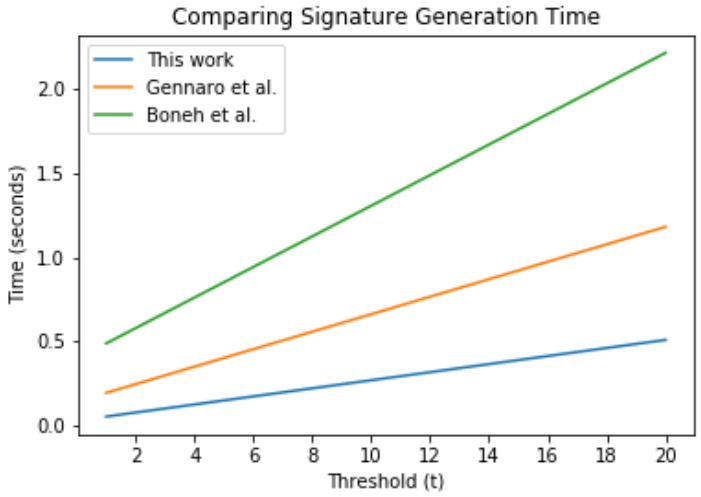


**Fig. 1. Comparison of the raw computation time as the threshold increases between this work and previous schemes.**

## 8 Conclusion

We have presented a threshold ECDSA protocol that is an improvement over the existing schemes by every metric. Although [17] has been available for some time, there are still to our knowledge no Bitcoin services or user wallets that offer threshold-signature security. We believe that this is due to the impracticality of their distributed key generation protocol. Having to rely on a trusted dealer to distribute key shares exposes a single point of failure for the system and in doing so runs contrary to the entire premise of using threshold signatures in the first place.

We solve this problem by presenting and implementing a new scheme with a highly efficient distributed key generation protocol. Together with our reduction in running time and data transferred, we believe that ECDSA threshold signatures are finally mature enough for adoption.

## 9 Acknowledgements

We thank Harry Kalodner, Yehuda Lindell, Ariel Nof, Ben Riva, and Omer Shlomovits for useful feedback and discussions and for pointing out errors in earlier versions.

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

## References

1. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Proceedings of the eighth annual ACM Symposium on Principles of distributed computing. pp. 201–209. ACM (1989)
2. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 480–494. Springer (1997)
3. Boneh, D.: Digital signature standard. In: Encyclopedia of cryptography and security, pp. 347–347. Springer (2011)
4. Boneh, D., Gennaro, R., Goldfeder, S.: Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security. In: Latincrypt (2017)

5. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 431–444. Springer (2000)
6. Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: Annual International Cryptology Conference. pp. 98–116. Springer (1999)
7. Canetti, R., Goldwasser, S.: An efficient *Threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In: Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. pp. 90–106 (1999)
8. Damgard, I., Groth, J.: Non-interactive and reusable non-malleable commitment schemes. In: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing. pp. 426–437. ACM (2003)
9. Damgård, I., Keller, M., Larraia, E., Miles, C., Smart, N.P.: Implementing aes via an actively/covertly secure dishonest-majority mpc protocol. In: International Conference on Security and Cryptography for Networks. pp. 241–263. Springer (2012)
10. Di Crescenzo, G., Ishai, Y., Ostrovsky, R.: Non-interactive and non-malleable commitment. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 141–150. ACM (1998)
11. Di Crescenzo, G., Katz, J., Ostrovsky, R., Smith, A.: Efficient and non-interactive non-malleable commitment. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 40–59. Springer (2001)
12. Doerner, J., Kondi, Y., Lee, E., et al.: Secure two-party threshold ecdsa from ecdsa assumptions. In: IEEE Symposium on Security and Privacy. p. 0. IEEE (2018)
13. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography,". In: Proceedings of the 23rd Annual Symposium on the Theory of Computing, ACM (1991)
14. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Annual International Cryptology Conference. pp. 16–30. Springer (1997)
15. Gennaro, R.: Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks. In: Annual International Cryptology Conference. pp. 220–236. Springer (2004)
16. Gennaro, R., Goldfeder, S.: Fast multiparty threshold ecdsa with fast trustless setup. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1179–1194. ACM (2018)
17. Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In: International Conference on Applied Cryptography and Network Security. pp. 156–174. Springer (2016)
18. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold dss signatures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 354–371. Springer (1996)
19. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold dss signatures. Information and Computation **164**(1), 54–84 (2001)
20. Gennaro, R., Micali, S.: Independent zero-knowledge sets. In: International Colloquium on Automata, Languages, and Programming. pp. 34–45. Springer (2006)
21. Gennaro, R., Micciancio, D., Rabin, T.: An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In: In Proc. of the 5th ACM Conference on Computer and Communications Security (CCS-98. Citeseer (1998)
22. Gilboa, N.: Two party rsa key generation. In: Advances in Cryptology - CRYPTO '99. pp. 116–129 (1999)
23. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing **17**(2), 281–308 (1988)
24. Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T.: Efficient rsa key generation and threshold paillier in the two-party setting. In: Cryptographers' Track at the RSA Conference. pp. 313–331. Springer (2012)
25. Jarecki, S., Lysyanskaya, A.: Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 221–242. Springer (2000)
26. Keller, M., Pastro, V., Rotaru, D.: Overdrive: making spdz great again. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 158–189. Springer (2018)
27. Kravitz, D.W.: Digital signature algorithm (Jul 27 1993), uS Patent 5,231,668
28. Lindell, Y.: Fast secure two-party ecdsa signing. In: Annual International Cryptology Conference. pp. 613–644. Springer (2017)

29. Lindell, Y., Nof, A.: Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1837–1854. ACM (2018)
30. MacKenzie, P., Reiter, M.K.: Two-party generation of dsa signatures. In: Annual International Cryptology Conference. pp. 137–154. Springer (2001)
31. MacKenzie, P., Yang, K.: On simulation-sound trapdoor commitments. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 382–400. Springer (2004)
32. Makriyannis, N., Peled, U.: A note on the security of GG18 (2021), Fireblocks Blog
33. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 223–238. Springer (1999)
34. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
35. Schnorr, C.: Efficient signature generation by smart cards. J. Cryptology **4**(3), 161–174 (1991)
36. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
37. Tymokhanov, D., Shlomovits, O.: Alpha-rays: Key extraction attacks on threshold ecdsa implementations. Cryptology ePrint Archive, Report 2021/1621 (2021), https://ia.cr/2021/1621

# A The ZK Proofs for the MtA protocol

In this section we describe the ZK proofs that are needed in the MtA protocol (see Section 3). The proofs are based on similar ones from [30]: specifically we prove statements that are simpler than the ones needed in [30].

In these proofs the Verifier uses an auxiliary RSA modulus $\tilde{N}$ which is the product of two safe primes $\tilde{P} = 2\tilde{p}+1$ and $\tilde{Q} = 2\tilde{q}+1$ with $\tilde{p}, \tilde{q}$ primes. The Verifier also uses two values $h_1, h_2 \in Z^*_{\tilde{N}}$ according to the commitment scheme in [14]. Security is based on the assumption that the Prover cannot solve the Strong RSA problem over $\tilde{N}$.

Therefore our initialization protocol must be augmented with each player $P_i$ generating an additional RSA modulus $\tilde{N}_i$, and values $h_{1i}, h_{2i}$, together with a proof that they are of the correct form (see [14]).

**Note:** The respondent proofs have been modified with respect to the previous version of this paper to address weaknesses pointed out in [37, 32].

## A.1 Range Proof

This proof is run by Alice (the initiator) in both MtA and MtAwc protocols.

The input for this proof is a Paillier public key $N, \Gamma$ and a value $c \in Z_{N^2}$. The prover knows $m \in Z_q$ and $r \in Z^*_N$ such that $c = \Gamma^m r^N \bmod N^2$, where $q$ is the order of the DSA group.

At the end of the protocol the Verifier is convinced that $m \in [-q^3, q^3]$.

- The Prover selects $\alpha \in_R Z_{q^3}$, $\beta \in_R Z^*_N$, $\gamma \in_R Z_{q^3\tilde{N}}$ and $\rho \in_R Z_{q\tilde{N}}$.
  The Prover computes $z = h_1^m h_2^\rho \bmod \tilde{N}$, $u = \Gamma^\alpha \beta^N \bmod N^2$, $w = h_1^\alpha h_2^\gamma \bmod \tilde{N}$.
  The Prover sends $z, u, w$ to the Verifier.
- The Verifier selects a challenge $e \in_R Z_q$ and sends it to the Prover.
- The Prover computes $s = r^e \beta \bmod N$, $s_1 = em + \alpha$ and $s_2 = e\rho + \gamma$ and sends $s, s_1, s_2$ to the Verifier.
- The Verifier checks that $s_1 \leq q^3$, $u = \Gamma^{s_1} s^N c^{-e} \bmod N^2$ and $h_1^{s_1} h_2^{s_2} z^{-e} = w \bmod \tilde{N}$.

Completeness. By inspection. Note that there is a negligible probability of failure for the honest prover (when $\alpha > q^3 - q^2$ – which happens with negligible probability – it might happen that $s_1 > q^3$).

Soundness. Let $\tilde{N}, \tilde{s}$ be our Strong RSA challenge. We show how to solve it using a Prover who succeeds on incorrect instances (i.e. where $|m| > q^3$).

Let $h_2 = \tilde{s}$ and $h_1 = h_2^{\chi}$ for a random $\chi \in Z_{q\tilde{N}}$. It is not hard to see that the distribution of these values is indistinguishable from the real one with sufficiently high probability.

Run the Prover on a successful execution over a challenge $e$ and then rewind him and find a successful execution with challenge $\hat{e}$. Therefore we have the same first message $z, u, w$ and two set of answers $s, s_1, s_2$ for challenge $e$, and $\hat{s}, \hat{s}_1, \hat{s}_2$ for challenge $\hat{e}$ both satisfying the verification equations. Let $\Delta_E = e - \hat{e}$, $\Delta_{s1} = s_1 - \hat{s}_1$ and $\Delta_{s2} = s_2 - \hat{s}_2$.

Let $\lambda = GCD(\Delta_{s2} + \chi\Delta_{s1}, \Delta_E)$. Assume $\lambda \neq \Delta_E$: denote with $\lambda_s = (\Delta_{s2} + \chi\Delta_{s1})/\lambda$ and $\lambda_E = \Delta_E/\lambda > 1$. Then we find $\mu, \nu$ such that $\mu\lambda_s + \nu\lambda_E = 1$.

Then the solution to the Strong RSA challenge is $\tilde{x} = z^{\mu}\tilde{s}^{\nu} \bmod \tilde{N}, \lambda_E$. Indeed note that

$$w = h_1^{s_1} h_2^{s_2} z^{-e} = h_1^{\hat{s}_1} h_2^{\hat{s}_2} z^{-\hat{e}} \bmod \tilde{N}$$

therefore

$$z^{\Delta_E} = h_1^{\Delta_{s1}} h_2^{\Delta_{s2}} = \tilde{s}^{\Delta_{s2} + \chi\Delta_{s1}} \bmod \tilde{N}$$

which implies

$$z^{\lambda_E} = \tilde{s}^{\lambda_S} \bmod \tilde{N}$$

Concluding

$$\tilde{s} = \tilde{s}^{\mu\lambda_s + \nu\lambda_E} = [z^{\mu}\tilde{s}^{\nu}]^{\lambda_E} \bmod \tilde{N}$$

We now need to prove that the case $\lambda = \Delta_E$ cannot happen with high probability.

Consider first the case $\lambda = \Delta_E$ but $\Delta_E$ does not divide $\Delta_{s1}$. Write $\chi = \chi_0 + \chi_1\tilde{p}\tilde{q}$ with $\chi_1$ chosen uniformly at random from a set of size $> q$. Note that the value $\chi_1$ is information theoretically secret from the adversary (who only has $h_1, h_2$). We have that

$$\Delta_{s2} + \chi\Delta_{s1} = \Delta_{s2} + \chi_0\Delta_{s1} + \chi_1\Delta_{s1}\tilde{p}\tilde{q}$$

Then there is a prime power $a^b$ (with $a \geq 2$) such that $a^b | \Delta_E$, $a^{b-1} | \Delta_{s1}$ but $a^b$ does not divide $\Delta_{s1}$. Note that this implies that $a^{b-1} | \Delta_{s2}$. Set $c_0 = (\Delta_{s2} + \chi_0\Delta_{s1})/a^{b-1}$ and $c_1 = \Delta_{s1}\tilde{p}\tilde{q}/a^{b-1}$. We have that $c_0 + \chi_1 c_1 = 0 \bmod a$ and $c_1 \neq 0 \bmod a$. The number of elements $\chi_1$ for which this equivalence holds is at most $q/a + 1$ and thus the probability of this holding for a random choice of $\chi_1$ is at most $\frac{1}{a} + \frac{1}{q}$ which is at most $\frac{1}{2} + \frac{1}{q}$. Otherwise we are in the case above with $\lambda \neq \Delta_E$.

Now consider the case $\lambda = \Delta_E$ and $\Delta_E | \Delta_{s1}$. Note that this implies that $\Delta_E | \Delta_{s2}$ as well. Define $m_1 = \Delta_{s1}/\Delta_E$, $\rho_1 = \Delta_{s2}/\Delta_E$, $\alpha_1 = (e\hat{s}_1 - \hat{e}s_1)/\Delta_E$, $\gamma_1 = (e\hat{s}_2 - \hat{e}s_2)/\Delta_E$.

These ensure that $z = h_1^{m_1} h_2^{\rho_1} \bmod \tilde{N}$, $w = h_1^{\alpha_1} h_2^{\gamma_1} \bmod \tilde{N}$ , $s_1 = em_1 + \alpha_1$ and $\hat{s}_1 = \hat{e}m_1 + \alpha_1$.

Finally denote with $m_1' = \Delta_{s1}\Delta_E^{-1} \bmod N$ and $\alpha_1' = (e\hat{s}_1 - \hat{e}s_1)\Delta_E^{-1} \bmod N$. Note that since $m_1' = m_1 \bmod N$ and $\alpha_1' = \alpha_1 \bmod N$, there must be $r_1, \beta' \in Z_N^*$ such that

$$c = \Gamma^{m_1'} r_1^N \quad \text{and} \quad u = \Gamma^{\alpha_1'} (\beta')^N \bmod N^2$$

At this point we know the following facts

$$s_1 < q^3 \quad s_1 = em_1 + \alpha_1 \quad s_1 = em_1' + \alpha_1 \bmod N$$

$$\hat{s}_1 < q^3 \quad \hat{s}_1 = \hat{e}m_1 + \alpha_1 \quad \hat{s}_1 = \hat{e}m_1' + \alpha_1 \bmod N$$

Therefore we can prove that $m_1 \in [-q^3, q^3]$ since $|m_1| \leq |\Delta_{s1}| \leq q^3$. But this implies that $m_1' \in [-q^3, q^3]$ since $m_1' = m_1 \bmod N$ and $N > q^7$.

Honest-Verifier Zero-Knowledge. The simulator proceeds as in [30]. Choose $z, s, s_1, s_2, e$ according to the appropriate distribution and set $u = \Gamma^{s_1} s^N c^{-e} \bmod N$ and $w = h_1^{s_1} h_2^{s_2} z^{-e} \bmod \tilde{N}$.

### A.2 Respondent ZK Proof for MtAwc

This proof is run by Bob (the responder) in the MtAwc protocol. For the MtA protocol a simpler version of this proof if needed, which we present later.

The input for this proof is a Paillier public key $N, \Gamma$ and two values $c_1, c_2 \in Z_{N^2}$, together with a value $X$ in $\mathcal{G}$ the DSA group.

The Prover knows $x \in Z_q$, $y \in Z_{q^5}$ and $r \in Z_N^*$ such that $c_2 = c_1^x \Gamma^y r^N \bmod N^2$, and $X = g^x \in \mathcal{G}$, where $q$ is the order of the DSA group.

At the end of the protocol the Verifier is convinced of the above and that $x \in [-q^3, q^3]$ and $y \in [-q^7, q^7]$.

- The Prover selects $\alpha \in_R Z_{q^3}$, $\rho \in_R Z_{q\tilde{N}}$, $\rho' \in_R Z_{q^3\tilde{N}}$, $\sigma \in Z_{q\tilde{N}}$, $\beta \in_R Z_N^*$, $\gamma \in_R Z_{q^7}$ and $\tau \in_R Z_{q^3\tilde{N}}$.

  The Prover computes $u = g^\alpha$, $z = h_1^x h_2^\rho \bmod \tilde{N}$, $z' = h_1^\alpha h_2^{\rho'} \bmod \tilde{N}$, $t = h_1^y h_2^\sigma \bmod \tilde{N}$, $v = c_1^\alpha \Gamma^\gamma \beta^N \bmod N^2$, and $w = h_1^\gamma h_2^\tau \bmod \tilde{N}$.

  The Prover sends $u, z.z', t, v, w$ to the Verifier.
- The Verifier selects a challenge $e \in_R Z_q$ and sends it to the Prover.
- The Prover computes $s = r^e \beta \bmod N$, $s_1 = ex + \alpha$, $s_2 = e\rho + \rho'$, $t_1 = ey + \gamma$ and $t_2 = e\sigma + \tau$.

  The Prover sends $s, s_1, s_2, t_1, t_2$ to the Verifier.
- The Verifier checks that $s_1 \leq q^3$, $t_1 \leq q^7$, $g^1 = X^e u \in \mathcal{G}$, $h_1^{s_1} h_2^{s_2} = z^e z' \bmod \tilde{N}$, $h_1^{t_1} h_2^{t_2} = t^e w \bmod \tilde{N}$, and $c_1^{s_1} s^N \Gamma^{t_1} = c_2^e v \bmod N^2$.

Completeness. By inspection (as before there is a negligible chance of failure for the honest prover when either $\alpha > q^3 - q^2$ or $\gamma > q^7 - q^6$

Soundness. Let $\tilde{N}, \tilde{s}$ be our Strong RSA challenge. We show how to solve it using a Prover who succeeds on incorrect instances (i.e. where $|x| > q^3$).

Let $h_2 = \tilde{s}$ and $h_1 = h_2^\chi$ for a random $\chi \in Z_{q\tilde{N}}$. It is not hard to see that the distribution of these values is indistinguishable from the real one with sufficiently high probability.

Run the prover on a successful execution over a challenge $e$ and then rewind him and find a successful execution with challenge $\hat{e}$. Therefore we have the same first message $u, z, z', t, v, w$ and two set of answers $s, s_1, s_2, t_1, t_2$ for challenge $e$, and $\hat{s}, \hat{s}_1, \hat{s}_2, \hat{t}_1, \hat{t}_2$ for challenge $\hat{e}$ both satisfing theverification equations. Let $\Delta_E = e - \hat{e}$, $\Delta_{s1} = s_1 - \hat{s}_1$, $\Delta_{s2} = s_2 - \hat{s}_2$, $\Delta_{t1} = t_1 - \hat{t}_1$ and $\Delta_{t2} = t_2 - \hat{t}_2$.

Let $\lambda = GCD(\Delta_{s2} + \chi\Delta_{s1}, \Delta_E)$. Assume $\lambda \neq \Delta_E$: denote with $\lambda_s = (\Delta_{s2} + \chi\Delta_{s1})/\lambda$ and $\lambda_E = \Delta_E/\lambda > 1$. Then we find $\mu, \nu$ such that $\mu\lambda_s + \nu\lambda_E = 1$.

Then the solution to the Strong RSA challenge is $\tilde{x} = z^\mu \tilde{s}^\nu \bmod \tilde{N}, \lambda_E$. Indeed note that

$$z' = h_1^{s_1} h_2^{s_2} z^{-e} = h_1^{\hat{s}_1} h_2^{\hat{s}_2} z^{-\hat{e}} \bmod \tilde{N}$$

therefore

$$z^{\Delta_E} = h_1^{\Delta_{s1}} h_2^{\Delta_{s2}} = \tilde{s}^{\Delta_{s2} + \chi\Delta_{s1}} \bmod \tilde{N}$$

which implies

$$z^{\lambda_E} = \tilde{s}^{\lambda_S} \bmod \tilde{N}$$

Concluding

$$\tilde{s} = \tilde{s}^{\mu\lambda_s + \nu\lambda_E} = [z^\mu \tilde{s}^\nu]^{\lambda_E} \bmod \tilde{N}$$

Let $\lambda' = GCD(\Delta_{t2} + \chi\Delta_{t1}, \Delta_E)$. In a similar way as above we can prove that if $\lambda' \neq \Delta_E$ then we can solve our Strong RSA challenge.

Therefore we can limit ourselves to the case $\lambda = \lambda' = \Delta_E$.

Consider first the case $\lambda = \lambda' = \Delta_E$ but $\Delta_E$ does not divide $\Delta_{s1}$. Write $\chi = \chi_0 + \chi_1 \tilde{p}\tilde{q}$ with $\chi_1$ chosen uniformly at random from a set of size $> q$. Note that the value $\chi_1$ is information theoretically secret from the adversary (who only has $h_1, h_2$). We have that

$$\Delta_{s2} + \chi\Delta_{s1} = \Delta_{s2} + \chi_0\Delta_{s1} + \chi_1\Delta_{s1}\tilde{p}\tilde{q}$$

Then there is a prime power $a^b$ (with $a \geq 2$) such that $a^b | \Delta_E$, $a^{b-1} | \Delta_{s1}$ but $a^b$ does not divide $\Delta_{s1}$. Note that this implies that $a^{b-1} | \Delta_{s2}$. Set $c_0 = (\Delta_{s2} + \chi_0\Delta_{s1})/a^{b-1}$ and $c_1 = \Delta_{s1}\tilde{p}\tilde{q}/a^{b-1}$. We have that $c_0 + \chi_1 c_1 = 0 \bmod a$ and $c_1 \neq 0 \bmod a$. The number of elements $\chi_1$ for which this equivalence holds is at most $q/a + 1$ and thus the probability of this holding for a random choice of $\chi_1$ is at most $\frac{1}{a} + \frac{1}{q}$ which is at most $\frac{1}{2} + \frac{1}{q}$. Otherwise we are in the case above with $\lambda \neq \Delta_E$.

In a similar fashion we can remove the case in which $\lambda = \lambda' = \Delta_E$ but $\Delta_E$ does not divide $\Delta_{t1}$.

Now consider the case $\lambda = \lambda' = \Delta_E$ with $\Delta_E | \Delta_{s1}$ and $\Delta_E | \Delta_{t1}$. Note that this implies that $\Delta_E | \Delta_{s2}$ and $\Delta_E | \Delta_{t2}$as well.

Define $x_1 = \Delta_{s1}/\Delta_E$, $\rho_1 = \Delta_{s2}/\Delta_E$, $\alpha_1 = (e\hat{s}_1 - \hat{e}s_1)/\Delta_E$, $\rho'_1 = (e\hat{s}_2 - \hat{e}s_2)/\Delta_E$, $y_1 = \Delta_{t1}/\Delta_E$, $\sigma_1 = \Delta_{t2}/\Delta_E$, $\gamma_1 = (e\hat{t}_1 - \hat{e}t_1)/\Delta_E$ and $\tau_1 = (e\hat{t}_2 - \hat{e}t_2)/\Delta_E$.

Define $x'_1 = x_1 \bmod N$ and $y'_1 = y_1 \bmod N$. Note that by definition

$$c_1^{x'_1} \Gamma^{y'_1} \kappa^N = c_2 \bmod N^2$$

for some $\kappa$ as needed. And $g^{x_1} = X \in \mathcal{G}$. So we have extracted the required $x, y$. As in the previous proof we can establish that $x_1, x'_1 \in [-q^3, q^3]$ and $y_1, y'_1 \in [-q^7, q^7]$.

Honest-Verifier Zero-Knowledge. The simulator proceeds as in [30] and in the previous ZK proof.

### A.3 Respondent ZK Proof for MtA

This proof is run by Bob (the responder) in the MtA protocol. It is a simpler version of the previous protocol where Bob only proves that $x, y$ are small (without proving that $x$ is the discrete log of any public value).

The input for this proof is a Paillier public key $N, \Gamma$ and two values $c_1, c_2 \in Z_{N^2}$.

The Prover knows $x \in Z_q$, $y \in Z_{q^5}$ and $r \in Z_N^*$ such that $c_2 = c_1^x \Gamma^y r^N \bmod N^2$ where $q$ is the order of the DSA group.

At the end of the protocol the Verifier is convinced of the above and that $x \in [-q^3, q^3]$ and $y \in [-q^7, q^7]$.

- The Prover selects $\alpha \in_R Z_{q^3}$, $\rho \in_R Z_{q\tilde{N}}$, $\rho' \in_R Z_{q^3\tilde{N}}$, $\sigma \in Z_{q\tilde{N}}$, $\beta \in_R Z_N^*$, $\gamma \in_R Z_{q^7}$ and $\tau \in_R Z_{q^3\tilde{N}}$.
  The Prover computes $z = h_1^x h_2^\rho \bmod \tilde{N}$, $z' = h_1^\alpha h_2^{\rho'} \bmod \tilde{N}$, $t = h_1^y h_2^\sigma \bmod \tilde{N}$, $v = c_1^\alpha \Gamma^\gamma \beta^N \bmod N^2$, and $w = h_1^\gamma h_2^\tau \bmod \tilde{N}$.
  The Prover sends $z, z', t, v, w$ to the Verifier.
- The Verifier selects a challenge $e \in_R Z_q$ and sends it to the Prover.
- The Prover computes $s = r^e \beta \bmod N$, $s_1 = ex + \alpha$, $s_2 = e\rho + \rho'$, $t_1 = ey + \gamma$ and $t_2 = e\sigma + \tau$.
  The Prover sends $s, s_1, s_2, t_1, t_2$ to the Verifier.
- The Verifier checks that $s_1 \leq q^3$, $t_1 \leq q^7$ $h_1^{s_1} h_2^{s_2} = z^e z' \bmod \tilde{N}$, $h_1^{t_1} h_2^{t_2} = t^e w \bmod \tilde{N}$, and $c_1^{s_1} s^N \Gamma^{t_1} = c_2^e v \bmod N^2$.

The proof is immediate from the previous one.

**Note about the previous version:** The respondent ZK proofs have been modified as follows. First of all note that now $y \in Z_{q^5}$ as required by the MtA protocols. This leads to the following changes in the ZK proof: $\gamma$ was chosen in $Z_N$ while now it is chosen in $Z_{q^7}$ and the verifier performs the additional range check $t_1 \leq q^7$ (no range check on $t_1$ was performed prior). Additionally there was a typo in the range of $\tau$ which was chosen in $Z_{q\tilde{N}}$ while now it is chosen in $Z_{q^3\tilde{N}}$.