Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security

Rosario Gennaro[1], Steven Goldfeder[2(✉)], and Arvind Narayanan[2]

[1] City College, City University of New York, New York, USA
rosario@cs.ccny.cuny.edu
[2] Princeton University, Princeton, USA
{stevenag,arvindn}@cs.princeton.edu

Abstract. While threshold signature schemes have been presented before, there has never been an optimal threshold signature algorithm for DSA. The properties of DSA make it quite challenging to build a threshold version. In this paper, we present a threshold DSA scheme that is efficient and optimal. We also present a compelling application to use our scheme: securing Bitcoin wallets. Bitcoin thefts are on the rise, and threshold DSA is necessary to secure Bitcoin wallets. Our scheme is the first general threshold DSA scheme that does not require an honest majority and is useful for securing Bitcoin wallets.

1 Introduction

Threshold signature schemes enable sharing signing power amongst n parties such that any subset of $t + 1$ can jointly sign, but any smaller subset cannot. This problem has received much attention in the cryptographic literature, and many such schemes have been designed. Some of these schemes produce signatures that are compatible with standard digital signature schemes. They replace only the signing algorithm and key generation algorithm, but the verification is compatible with the centralized signature schemes.

The Digital Signature Algorithm (DSA) is a very popular signature scheme, and a considerable amount of work has been done to build a threshold signing algorithm to produce a standard DSA signature. However, for reasons that we will elaborate in Sect. 4.2, building a threshold variant of DSA proved to be significantly difficult. While such schemes have been presented (e.g. [22,23,31]), they have serious drawbacks that make them unusable in practice: in particular no general scheme with an optimal number of servers is known. For the past 15 years, the problem has been mostly abandoned. The reason is twofold:

– As we discuss in Sect. 4.2 the technical difficulties in building a threshold-optimal variant of distributed DSA made this a challenging problem and it was not clear how to proceed from the solutions in [22,23,31].

M. Manulis et al. (Eds.): ACNS 2016, LNCS 9696, pp. 156–174, 2016.
DOI: 10.1007/978-3-319-39555-5_9

– There was never a pressing motivation to devise a solution for threshold DSA. Since there are optimal threshold schemes for other signature algorithms, one could choose a different scheme that was well suited for the problem at hand.

In recent years, a major application for threshold DSA signatures has arisen in the world of Bitcoin. Without a DSA/ECDSA threshold scheme, bitcoins are subject to a single point of failure and the risks of holding bitcoins are catastrophic. Motivated by this application, we tackle the technical challenges of threshold DSA, and present an efficient and optimal scheme.

The Motivation: Bitcoin's Security Conundrum. Bitcoin is a cryptographic e-cash system, by far the most widely used today. Unlike traditional banking transactions, Bitcoin transactions of any size can be fully automated – authorized only with a ECDSA signature. One's bitcoins are only as secure as the ECDSA key that can authorize their transfer; if this key is compromised, the bitcoins will be stolen. Unlike traditional banking transactions, once a Bitcoin transaction is enacted it is irreversible. Even if the coins are known to have been stolen, there is simply no way to reverse the offending transaction.

Indeed, the Bitcoin ecosystem is plagued by constant thefts. The statistics on Bitcoin hacks, thefts, and losses are extraordinary — there have been ten thefts of over 10,000 BTC each since mid-2011, and at least another thirty-four of over 1,000 BTC.[1] [4]. Kaspersky labs report detecting about a million infections per month of malware designed to search for and steal bitcoins [30].

The pervasiveness and regularity of these vulnerabilities highlight how Bitcoin is inherently theft-prone. For Bitcoin and cryptocurrencies to gain mainstream adoption, the current situation where a single rogue employee or a piece of malware can empty an organization's funds in hot storage instantly, irreversibly, and anonymously is simply untenable. Securing Bitcoin is equivalent to securing the keys that can authorize transactions. Instead of storing keys in a single location, keys should be split and signing should be authorized by a threshold set of computers. A breach of any number of machines up to the threshold will not allow the attacker to steal any money or glean any information about the key.

Since Bitcoin transactions use ECDSA keys, the only way to achieve this joint control is with an ECDSA threshold signature algorithm. While Bitcoin does have a built in "multi-signature" function for splitting control, using this severely compromises the confidentiality and anonymity of the participants as we explain in the full version of this paper.

Our Contributions. With a strong motivation for threshold DSA, we still lacked a scheme that was usable to secure Bitcoin keys. The best threshold signature scheme presented was that by Gennaro *et al.* [22]. That scheme, however, has a considerable setback. The key is distributed among n players such that a

[1] The majority, but not all, of these losses have been due to theft of keys.

group of size $t+1$ can jointly reconstruct the key. Yet, in order to produce a signature using their algorithm (without reconstructing the key), the participation of $2t+1$ players is required.[2]

This property of the scheme in [22] has various implications. First, requiring $n \geq 2t+1$ is very limiting in practice: for example it rules out an n-of-n sharing. Furthermore, the implications for a Bitcoin company that wants to distribute its signing power are severe. If the company chooses a threshold of t, then an attacker who compromises $t+1$ servers can steal all of the company's money. Yet, in order for the company to sign a transaction, they must set up $2t+1$ servers. In effect, they must double the number of servers, which makes the job of the attacker easier (as there are more servers for them to target).

Mackenzie and Reiter built a specialized scheme for the 2-of-2 signature case [31]. Yet no general DSA threshold scheme existed that did not suffer from these setbacks. In the full version of this paper, we sketch how to extend Mackenzie and Reiter to the multiparty case. While the extension does allow $t+1$ players to sign, it is quite inefficient as it requires $3t-1$ rounds of interaction, and the computation time and the storage grow with the number of players.

In this paper, we present a scheme that is both threshold-optimal and efficient. In particular:

1. It requires only $n \geq t+1$ servers.
2. The protocol requires only a constant number of rounds.
3. The computation time for each player is constant.[3]
4. Players require only a constant amount of storage.

Our scheme is practical and efficient. We implemented it and evaluated it, and it is the only scheme that is fully compatible with Bitcoin and efficient enough to now be a true candidate for any use case where a threshold signature scheme is desired. We have also spoken with various Bitcoin companies who confirmed that they are eager to incorporate our protocol to secure their systems.

2 Model, Definitions and Tools

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

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

[2] We note that throughout this paper we use the (t, n) notation consistently with how it's used in previous threshold signature works. In particular, a (t, n) signing scheme is secure against t colluding players and requires at least $t+1$ participants. In the Bitcoin multisignature notation, however, t-of-n refers to a scheme which is secure against $t-1$ malicious players and requires t participants to sign.

[3] That is to compute the players share, the computation time does not grow with the number of players. Players do however need to verify proofs from all players.

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

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

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

Adversaries can also be categorized as *static* or *adaptive*. A static adversary chooses the corrupted players at the beginning of the protocol, while an adaptive one chooses them during the computation. In the following, for simplicity, we assume the adversary to be static, though the techniques from [13,28] can be used to extend our result to the adaptive adversary case.

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

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

The notion of security for signature schemes was formally defined in [25] in various flavors. The following definition captures the strongest of these notions: existential unforgeability against adaptively chosen message attack.

Definition 1. *We say that a signature scheme* $\mathcal{S} = (\mathsf{Key\text{-}Gen}, \mathsf{Sig}, \mathsf{Ver})$ *is* unforgeable *if no adversary who is given the public key* y *generated by* Key-Gen*, and the signatures of* k *messages* $m_1, \ldots, m_k$ *adaptively chosen, can produce the signature on a new message* m *(i.e.,* $m \notin \{m_1, \ldots, m_k\}$*) with non-negligible (in* λ*) probability.*

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

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

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

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

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

Secure Threshold Signature Schemes

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

This is analogous to the notion of existential unforgeability under chosen message attack as defined by Goldwasser, Micali, and Rivest [25]. Notice that now the adversary does not just see the signatures of k messages adaptively chosen, but also the internal state of the corrupted players and the public communication of the protocols. Following [25] one can also define weaker notions of unforgeability.

In order to prove unforgeability, we use the concept of *simulatable adversary view* [12,26]. Intuitively, this means that the adversary who sees all the information of the corrupted players and the signature of m, could generate by itself all the other public information produced by the protocol Thresh-Sig. This ensures that the run of the protocol provides no useful information to the adversary other than the final signature on m.

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

1. *The protocol* Thresh-Key-Gen *is simulatable. That is, there exists a simulator SIM_1 that, on input a public key y, can simulate the view of the adversary on an execution of* Thresh-Key-Gen *that results in y as the public output.*
2. *The protocol* Thresh-Sig *is simulatable. That is, there exists a simulator SIM_2 that, on input the public input of* Thresh-Sig *(in particular the public key y and*

the message m), t shares $x_{i_1}, \ldots, x_{i_t}$, and a signature sig of m, can simulate the view of the adversary on an execution of Thresh-Sig *that generates sig as an output.*

Threshold Optimality. Given a (t, n)-threshold signature scheme, obviously $t + 1$ honest players are necessary to generate signatures. We say that a scheme is *threshold-optimal* if $t + 1$ honest players also suffice.

The main contribution of our work is to present a threshold-optimal DSA scheme for general t. The only known optimal scheme was in [31] for the case of $(1, 2)$-threshold (i.e. 2-out-of-2) threshold DSA. The protocol in [22,23] is not threshold-optimal as it requires $2t + 1$ honest players to compute a signature.

We point out that if we consider an honest-but-curious adversary, then it will suffice to have $n = t + 1$ players in the network to generate signatures (since all players will behave honestly, even the corrupted ones). But in the presence of a malicious adversary one needs at least $n = 2t + 1$ players in total to guarantee *robustness*, i.e. the ability to generate signatures even in the presence of malicious faults. In that sense our protocol improves over [22,23] where $n = 3t + 1$ players are required to guarantee robustness.

But we want to minimize the number of servers, and keep it at $n = t + 1$ even in the presence of malicious faults. In this case we give up on robustness, meaning that we cannot guarantee anymore that signatures will be provided. But we can still prove that our scheme is unforgeable. In other words, an adversary who corrupts almost all the players in the network can only create a denial of service attack, but not learn any information that would allow him to forge. This is another contribution of our paper, since it is not clear how to provide such "dishonest majority" analysis in the case of [22,23].

2.1 Additively Homomorphic Encryption

We assume the existence of an encryption scheme E which is additively homomorphic modulo a large integer N: i.e. given $\alpha = E(a)$ and $\beta = E(b)$, where $a, b \in Z_N$, there is an efficiently computable operation $+_E$ over the ciphertext space such that

$$\alpha +_E \beta = E(a + b \bmod N)$$

Note that if x is an integer, given $\alpha = E(a)$ we can also compute $E(xa \bmod N)$ efficiently. We refer to this operation as $x \times_E \alpha$. We denote the message space of E by $\mathcal{M}_E$ and the ciphertext space by $\mathcal{C}_E$.

With $\bigoplus_{i=1}^{t+1} \alpha_i$ we denote the summation over the addition operation $+_E$ of the encryption scheme: i.e. $\bigoplus_{i=1}^{t+1} \alpha_i = \alpha_1 +_E \ldots +_E \alpha_{t+1}$.

One instantiation of a scheme with these properties is Paillier's encryption scheme [35]. We recall the details of the scheme here.

- Key Generation: generate two large primes P, Q of equal length. and set $N = PQ$. Let $\lambda(N) = lcm(P - 1, Q - 1)$ be the Carmichael function of N. Finally choose $\Gamma \in Z^*_{N^2}$ such that its order is a multiple of N. The public key is (N, Γ) and the secret key is $\lambda(N)$.

- Encryption: to encrypt a message $m \in Z_N$, select $x \in_R Z_N^*$ and return $c = \Gamma^m x^N \bmod N^2$.
- Decryption: to decrypt a ciphertext $c \in Z_{N^2}$, let L be a function defined over the set $\{u \in Z_{N^2} : u = 1 \bmod N\}$ computed as $L(u) = (u-1)/N$. Then the decryption of c is computed as $L(c^{\lambda(N)})/L(\Gamma^{\lambda(N)}) \bmod N$.
- Homomorphic Properties: Given two ciphertexts $c_1, c_2 \in Z_{N^2}$ define $c_1 +_E c_2 = c_1 c_2 \bmod N^2$. If $c_i = E(m_i)$ then $c_1 +_E c_2 = E(m_1 + m_2 \bmod N)$. Similarly, given a ciphertext $c = E(m) \in Z_{N^2}$ and a number $a \in Z_n$ we have that $a \times_E c = c^a \bmod N^2 = E(am \bmod N)$.

2.2 Threshold Cryptosystems

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

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

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

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

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

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

We point out that threshold variations of Paillier's scheme have been presented in the literature [2,15,16,27]. In order to instantiate our dealerless protocol, we must use the scheme from [27] as it is the only one that includes a dealerless key generation protocol that does not require $n \geq 2t + 1$.

3 Independent Trapdoor Commitments

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

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

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

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

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

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

Such a commitment is *non-malleable* [19] if no adversary $\mathcal{A}$, given a commitment C to a message 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 [17]). We are going to use a related property called *independence* and introduced in [24].

Consider the following scenario: an honest party produces a commitment C and the adversary, after seeing C, will produce another commitment C'

(which we require to be different from C in order to prevent the adversary from simply copying the behavior of the honest party and outputting an identical committed value). At this point the value committed by the adversary should be *fixed*, i.e. no matter how the honest party open his commitment the adversary will always open in a unique way.

The following definition takes into account that the adversary may see and output many commitments [14].

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

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

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

Candidate Independent Trapdoor Commitments. As shown in [24] independence implies non-malleability. We point out that *all* non-malleable commitments in the literature are also independent ones.

The non-malleable commitment schemes in [17,18] 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 [14,21,32]), and all these schemes can also be proven independent according to the definition presented above.

4 The Digital Signature Standard

We define a generic G-DSA signature algorithm as follows. The public parameters include a cyclic group $\mathcal{G}$ of prime order q generated by an element g, a hash function H defined from arbitrary strings into Z_q, and another hash function H' defined from $\mathcal{G}$ to Z_q.

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

– Verification Algorithm On input $M, (r, s)$ and y, the receiver checks that $r, s \in Z_q$ and computes

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

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

The traditional DSA algorithm is obtained by choosing large primes p, q such that $q|(p-1)$ and setting $\mathcal{G}$ to be the subgroup of Z_p^* of order q. 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.

4.1 Threshold DSA

As discussed in Sect. 2, in a (t, n)-threshold signature scheme the secret key is shared among n servers, in such a way that any t of them has no information about the secret key, while n players can sign a message using a communication protocol that does not require the secret key to be reconstructed. A scheme is threshold-optimal if exactly $n = t + 1$ honest players can sign.

For the case of DSA, in [22,23] Gennaro *et al.* present such a non-optimal scheme that requires $n = 2t + 1$ honest players to participate in a signature. In particular this prevents the classical "2-out-of-2" case where the key is split among 2 servers so that both must cooperate to sign, while 1 has no information about the secret key (in [22,23] if 1 server has no information about the key, then one would need 3 servers to sign). The 2-out-of-2 case is handled by [31].

The schemes of [22,31] are described for the specific case of the DSA scheme, but it is not hard to see that they both work for the generic G-DSA scheme, and therefore also for ECDSA. We present our scheme with the G-DSA notation.

4.2 The Technical Issues

The main technical issue in constructing threshold DSA signatures is dealing with the fact that *both* the secret key x and the nonce k have to remain secret. This means that in a threshold scheme, they must be shared among the servers. The protocol in [22] is based on Shamir's secret sharing [39], which means that both x and k are shared using polynomials of degree t. Due to the fact that k and x are multiplied to compute s, the end result is that s will be shared using a polynomial of degree $2t$, which requires $2t + 1$ honest players to reconstruct.

The protocol in [31] gets around the above problem by using a multiplicative sharing of the secret values in the protocol. This allows an efficient way to multiply k and x without incurring an increase of the number of players required to reconstruct. However it only works for 2 players.

Our first approach was to first extend the techniques in [31] to the case of t-out-of-t players, but that required $O(t)$ rounds and the use of Paillier's encryption scheme with a modulus $N = O(q^{3t-1})$. Moreover if one wanted to extend that to a t-out-of-n scheme using a combinatorial structure, it would require $O(n^t)$ storage, making it feasible only for small values of n and t. For comparison, we have included this scheme in the full version of this paper.

The scheme we present in the next section requires only 6 rounds, constant amount of storage from the players, and uses a Paillier modulus $N > q^8$.

5 Our Scheme

In this section, we describe our scheme in three parts. First we describe the initialization phase, in which some common parameters are chosen. Then we describe the key generation protocol, in which the parties jointly generate a DSA key pair $(x, y = g^x)$ with y public and x shared among the players. Finally, we describe the signature generation protocol.

In the following, we assume that if any player does not perform according to the protocol (e.g. by failing a ZK proof, or refusing to open a committed value), then the protocol aborts and stops.

5.1 Initialization Phase

In this phase, a common reference string containing the public information pk for an independent trapdoor commitment KG, Com, Ver, Equiv is selected and published. This could be accomplished by a trusted third party, who can be assumed to erase any secret information (i.e. the trapdoor of the commitment).[4] The common parameters $\mathcal{G}, g, q$ for the DSA scheme are assumed to be known.

5.2 Key Generation Protocol

Here we describe how the players can jointly generate a DSA key pair $(x, y = g^x)$ with y public and x shared among the players. The idea is to generate a public key E for an additively (mod N) homomorphic encryption scheme E, together with the secret key D in shared form among the players. The value N is chosen to be larger than q^8. Then a value x is generated, and encrypted with E, with the value $\alpha = E(x)$ made public. Note that this is an implicit (t, n) secret sharing of x, since the decryption key of E is shared among the players. We use independent trapdoor commitments KG, Com, Ver, Equiv to enforce the independence of the values contributed by each player to the selection of x (in the following, for simplicity we may drop the public key pk and the randomness input when describing the computation of a commitment and write $[C, D] = \mathsf{Com}(m)$)

[4] Another option is to use a publicly verifiable method that generates the public information, without the trapdoor being known. For example the public parameters in [18] could be generated by using a "random oracle" over some public information without anybody knowing the trapdoor.

More specifically, the scheme is described below. We assume that if any commitment opens to $\perp$ or if any of the ZK proofs fail, the protocol terminates without an output.

- The parties run the key generation protocol $\mathsf{TEnc\text{-}Key\text{-}Gen}$ for an additively homomorphic encryption scheme E. If using Paillier's encryption scheme, we can use the threshold version from [27] with $N > q^8$.
- Each player P_i selects a random value $x_i \in Z_q$, computes $y_i = g^{x_i} \in \mathcal{G}$ and $[C_i, D_i] = \mathsf{Com}(y_i)$;
- Each player P_i broadcasts C_i
- Each player P_i broadcasts
  - D_i which allows everybody to compute $y_i = \mathsf{Ver}(C_i, D_i)$.
  - $\alpha_i = E(x_i)$;
  - a ZK argument Π_i that states
    * $\exists\ \eta \in [-q^3, q^3]$ such that
    * $g^\eta = y_i$
    * $D(\alpha_i) = \eta$

    If any of the ZK arguments fails, the protocol terminates.
- The players compute $\alpha = \bigoplus_{i=1}^{t+1} \alpha_i$ and $y = \prod_{i=1}^{t+1} y_i$.

The public key for the DSA is set to y. We note that $y = g^x$ and that $\alpha = E(x')$ with $x' = x \bmod q$ since $x' = \sum_{i=1}^{t+1} x_i$ is computed modulo N, but since $N > q^8$, we have that x' is computed actually over the integers.

5.3 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. Here too, we assume that if any commitment opens to $\perp$ or if any of the ZK proofs fail, the protocol terminates without an output.

- Round 1

  Each player P_i
  - chooses $\rho_i \in_R Z_q$
  - computes $u_i = E(\rho_i)$ and $v_i = \rho_i \times_E \alpha = E(\rho_i x)$
  - computes $[C_{1,i}, D_{1,i}] = \mathsf{Com}([u_i, v_i])$ and broadcasts $C_{1,i}$
- Round 2

  Each player P_i broadcasts
  - $D_{1,i}$. This allows everybody to compute $[u_i, v_i] = \mathsf{Ver}(C_{1,i}, D_{1,i})$
  - a zero-knowledge argument $\Pi_{(1,i)}$ which states
    * $\exists\ \eta \in [-q^3, q^3]$ such that
    * $D(u_i) = \eta$
    * $D(v_i) = \eta D(E(x))$

  Players compute $u = \bigoplus_{i=1}^{t+1} u_i = E(\rho)$ and $v = \bigoplus_{i=1}^{t+1} v_i = E(\rho x)$, where $\rho = \sum_{i=1}^{t+1} \rho_i$ (over the integers)
- Round 3

  Each player P_i

- chooses $k_i \in_R Z_q$ and $c_i \in_R [-q^6, q^6]$
- computes $r_i = g^{k_i}$ and $w_i = (k_i \times_E u) +_E E(c_i q) = E(k_i \rho + c_i q)$
- computes $[C_{2,i}, D_{2,i}] = \mathsf{Com}(r_i, w_i)$ and broadcasts $C_{2,i}$

– Round 4

Each player P_i broadcasts

- $D_{2,i}$ which allows everybody to compute $[r_i, w_i] = \mathsf{Ver}(C_{2,i}, D_{2,i})$
- a zero-knowledge argument $\Pi_{(2,i)}$ which states
  * $\exists\ \eta \in [-q^3, q^3]$ such that
  * $g^\eta = r_i$
  * $D(w_i) = \eta D(u) \bmod q$

Players compute $w = \bigoplus_1^{t+1} w_i = E(k\rho + cq)$ where $k = \sum_{i=1}^{t+1} k_i$ and $c = \sum_{i=1}^{t+1} c_i$(over the integers). Players also compute $R = \Pi_1^{t+1} r_i = g^k$ and $r = H'(R) \in Z_q$

– Round 5

- players jointly decrypt w using TDec to learn the value $\eta \in [-q^7, q^7]$ such that $\eta = k\rho \bmod q$ and $\psi = \eta^{-1} \bmod q$
- Each player computes

$$\begin{aligned}\sigma &= \psi \times_E [(m \times_E u) +_E (r \times_E v)] \\ &= \psi \times_E [E(m\rho) +_E E(r\rho x)] \\ &= (k^{-1}\rho^{-1}) \times_E [E(\rho(m + xr))] \\ &= E(k^{-1}(m + xr)) \\ &= E(s)\end{aligned}$$

– Round 6

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

Remark: The Size of the Modulus N**.** We note that in order for the protocol to be correct, all the homomorphic operations over the ciphertexts (which are modulo N), must not "conflict" with the operations modulo q of the DSA algorithms. We note that the values encrypted under E are $\sim q^7$. Indeed the ZK proofs guarantee that the values $k, \rho < q^3$. Moreover the "masking" value cq in the decryption of η is at most q^7, so the encrypted values in w_i are never larger than q^8. By choosing $N > q^8$ we guarantee that when we manipulate ciphertexts, all the operations on the plaintexts happen basically over the integers, without taking any modular reduction mod N.

Remark: The Distribution of Signatures. The adversary has the ability to affect the distribution of signatures by refusing to open his commitment if the signature is not to his liking. This leads to a possible loss of anonymity as an adversary can alter the distribution of the signatures in such a way that helps one distinguish the signatures made by this group.

In practice this is not a problem for Bitcoin. The only way one can affect distribution is by refusing to open, and if one of the players refused to open, we would detect that they are corrupted and reboot them. In the full version of this

paper we analyze the anonymity lost under these assumptions. The anonymity in Bitcoin is linear in the number of users, and we find that the most the adversary can do is reduce the anonymity equivalently to halving the number of users.

5.4 Security Proof and Zero-Knowledge Arguments

A simulation based security proof and the details of how to implement the zero-knowledge proofs are presented in the full version of this paper.

6 Threshold Security for Bitcoin Wallets

In this section, we give an overview of Bitcoin, discuss the threat model, and show how threshold signatures provide a solution for the most pressing threats.

6.1 Bitcoin

Bitcoin is a decentralized digital currency [34]. Bitcoins are owned by *addresses*; an address is simply the hash of a public key. To transfer bitcoins from one address to another, a *transaction* is constructed that specifies one or more input addresses from which the funds are to be debited, and one or more output addresses to which the funds are to be credited. For each input address, the transaction contains a reference to a previous transaction which contained this address as an output address. In order for the transaction to be valid, it must be signed by the private key associated with each input address, and the funds in the referenced transactions must not have already been spent [6,34].

Each output of a transaction may only be referenced as the input to a single subsequent transaction. It is thus necessary to spend the entire output at once. It is often the case that one only wishes to spend part of an output that was received in a previous transaction. This is accomplished by means of a *change address* where one lists their own address as an output of the transaction.

While the sender could include their input address as the change address in the output, the best and recommended practice is to send the change to a newly generated addresses. The motivation for doing so is increased anonymity as it makes it harder to track which addresses are owned by which individuals.

A Bitcoin *wallet* is a software abstraction which seamlessly manages multiple addresses on behalf of a user. Users do not deal with the low level details of their addresses. They just see their total balance, and when they want to send bitcoins to another address, they specify the amount to be transferred. The wallet software chooses the input and change addresses and constructs the transaction.

Signed transactions are broadcast to the Bitcoin peer-to-peer network. They are validated by *miners* who group transactions together into *blocks*. Miners participate in a distributed consensus protocol that collects these blocks into an append-only global log called the *block chain*.

While the original Bitcoin paper does not specify which signature algorithm to use, the current implementation uses ECDSA over the secp256k1 curve [6–8].

6.2 Threat Model

To classify the problems, we distinguish between internal and external threats as well as between hot and cold wallets. While the term wallet is generally used loosely to refer to a software abstraction (as described in the previous section), we will use the term in the rest of the paper in a more precise sense.

Definition 4 (wallet). *A collection of addresses with the same security policy together with a software program or protocol that allows spending from those addresses in accordance with that policy.*

"Security policy" encompasses the ownership or access-control list and the conditions under which bitcoins in the wallet may be spent.

The terms *hot wallet* and *cold wallet* derive from the more general terms *hot storage*, meaning online storage, and *cold storage*, meaning offline storage.

Definition 5 (Hot wallet/Cold wallet). *A* hot wallet *is a wallet from which bitcoins can be spent without accessing cold storage. Conversely, a* cold wallet *is a wallet from which bitcoins cannot be spent without accessing cold storage.*

Note that these new definitions refer to the desired effect, not the method of achieving it. The desired effect of a business that maintains a hot wallet is the ability to spend bitcoins online without having to access cold storage.

Table 1. Taxonomy of threats

Adversary	Hot wallet	Cold wallet
Insider	Vulnerable by default; our methods are necessary	Reduces to physical security by default; our methods can help
External (network)	Reduces to network security by default; our methods can help	Safe

Table 1 shows four types of possible threats to Bitcoin wallets. Securing a cold wallet is a physical security problem. While a network adversary is unable to get to a cold wallet, traditional physical security measures can be used to protect it from insiders — for example, private keys printed on paper and stored in a locked safe with video surveillance.

In addition, our methods may be used to supplement physical security measures. Instead of storing the key in a single location, the business can store shares of the key in different locations. The adversary will thus have to compromise security in multiple locations in order to recover the key. Indeed, this is one use case where Bitcoin companies have expressed great interest in implementing our threshold signature scheme.

Protecting hot wallets from internal attackers is the most pressing problem. Our central claim is that due to the irreversibility of Bitcoin transactions, the level of insecurity of this threat category has no parallels in traditional finance or network security, necessitating Bitcoin-specific solutions. Whereas traditional banking systems can incorporate detection and recovery in their security measures, Bitcoin security must come from prevention; the irreversibility precludes any recovery options.

7 Implementation and Evaluation

In this section, we describe and evaluate our implementation of our protocol. We describe two different parts of our implementation. We also created a desktop Bitcoin wallet and android app that used threshold signatures for two factor authentication. The implementation details of the two factor applications are in the full version of this paper.

7.1 Our Protocol

We fully implemented our protocol in Java. We chose Java since it works well cross-platform, and in particular since it can run on mobile devices. We wrote our code as library functions that are easily called and incorporated into company's existing codebase.

We have released code for our threshold signing protocol[5] as well as for the two factor wallet[6]. We plan to update the repository shortly with code for the dealerless key generation protocol.

For the independent trapdoor commitment scheme, we implemented the second protocol in [21]. This protocol uses pairing based cryptography, and we used the Jpair library to facilitate this. For the underlying Paillier scheme, we use a modified version of the Java implementation of threshold Paillier in [36],

7.2 Runtime

We benchmarked our implementation on a machine with a 2.4 GHz Intel Core i7 Processor. Our implementation was extremely efficient. The time to generate a signature is a function of t, the number of players actively involved. We note that in the protocol, the time for each player to generate their shares is not affected by the number of other participants. However, players need to verify zero knowledge proofs and check commitments from all other players, and thus the runtime is affected by the number of players.

Our evaluation found that without checking proofs or commitments, each players runtime is 1.1 s. Checking proofs and commitments takes 0.5 s per player, and the runtime R in seconds is thus given by the following:

[5] https://github.com/citp/ThresholdECDSA.
[6] https://github.com/citp/TwoFactorBtcWallet.

$$R(t) = 1.1 + 0.5t$$

Not counting for network latency, this means if 3 participants are required to generate a signature (so $t = 2$), it takes 2.1 s to generate a signature.

8 Conclusion

In this paper, we presented the first threshold-optimal signature scheme for DSA. We proved its security, implemented it, and evaluated it. Our scheme is quite efficient, and our implementation confirms that this scheme is ready to be used. Indeed, many Bitcoin companies have expressed great interest in our scheme as it provides a much needed solution to Bitcoin's security problem. We have open sourced our two-factor app and our general (t, n) signature code as well so that companies can benefit from our results and begin to use them immediately.

Acknowledgements. We would like to thank Dan Boneh, Joseph Bonneau, Edward W. Felten, Harry Kalodner, and Joshua Kroll for helpful input and feedback. We would also like to thank Harry Kalodner for his work in implementing the two factor wallet. We would like to thank Daniel Wichs for raising the question of how the adversary's ability to alter the signature distribution affects anonymity.

Rosario Gennaro is supported by NSF Grant 1545759. Steven Goldfeder is supported by the National Science Foundation Graduate Research Fellowship under grant number DGE 1148900. Arvind Narayanan is supported by NSF Grant CNS-1421689.

References

1. Andresen, G.: Github: Shared Wallets Design. https://gist.github.com/gavinandresen/4039433
2. Baudron, O., Fouque, P.-A., Pointcheval, D., Poupard, G., Stern, J.: Practical multi-candidate election system. In: PODC 2001
3. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (1997)
4. Bitcoin Forum member dree12, List of Bitcoin Heists (2013). https://bitcointalk.org/index.php?topic=83794.0
5. Bitcoin Forum member gmaxwell, Coinjoin: Bitcoin privacy in the real world (2013). https://bitcointalk.org/index.php?topic=279249.0
6. Bitcoin wiki: Transactions. https://en.bitcoin.it/wiki/Transactions
7. Bitcoin wiki: Elliptic Curve Digital Signature Algorithm. https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm
8. Bitcoin wiki: Secp256k1. https://en.bitcoin.it/w/index.php?title=Secp256k1
9. Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J.A., Felten, E.W.: Mixcoin: anonymity for bitcoin with accountable mixes. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 486–504. Springer, Heidelberg (2014)
10. Camenisch, J., Kiayias, A., Yung, M.: On the portability of generalized schnorr proofs. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 425–442. Springer, Heidelberg (2009)

11. Camenisch, J., Krenn, S., Shoup, V.: A framework for practical universally composable zero-knowledge protocols. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 449–467. Springer, Heidelberg (2011)
12. Canetti, R., Security, U.C.: A new paradigm for cryptographic protocols. In: Proceedings of 42nd IEEE Symposium on Foundations of Computer Science (FOCS 2001) (2001)
13. Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 98–116. Springer, Heidelberg (1999)
14. Damgård, I., Groth, J.: Non-interactive and reusable non-malleable commitment schemes. In: Proceedings of 35th ACM Symposium on Theory of Computing (STOC 2003) (2003)
15. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001)
16. Damgård, I.B., Koprowski, M.: Practical threshold RSA signatures without a trusted dealer. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 152–165. Springer, Heidelberg (2001)
17. Di Crescenzo, G., Ishai, Y., Ostrovsky, R.: Non-interactive and non-malleable commitment. In: Proceedings of 30th ACM Symposium on Theory of Computing (STOC 1998) (1998)
18. Di Crescenzo, G., Katz, J., Ostrovsky, R., Smith, A.: Efficient and non-interactive non-malleable commitment. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 40–59. Springer, Heidelberg (2001)
19. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography. SIAM J. Comp. **30**(2), 391–437 (2000)
20. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (1997)
21. Gennaro, R.: Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 220–236. Springer, Heidelberg (2004)
22. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold DSS signatures. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 354–371. Springer, Heidelberg (1996)
23. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 295–310. Springer, Heidelberg (1999)
24. Gennaro, R., Micali, S.: Independent zero-knowledge sets. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 34–45. Springer, Heidelberg (2006)
25. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988)
26. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. SIAM. J. Comput. **18**(1), 186–208 (1989)
27. Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T.: Efficient RSA key generation and threshold paillier in the two-party setting. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 313–331. Springer, Heidelberg (2012)
28. Jarecki, S., Lysyanskaya, A.: Adaptively secure threshold cryptography: introducing concurrency, removing erasures (Extended Abstract). In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 221–242. Springer, Heidelberg (2000)

29. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Secur. **1**(1), 36–63 (2001)
30. Kaspersky Labs, Financial cyber threats in: 2013. Part 2: malware (2013). http://securelist.com/analysis/kaspersky-security-bulletin/59414/financial-cyber-threats-in-2013-part-2-malware/
31. MacKenzie, P., Reiter, M.: Two-party generation of DSA signatures. Int. J. Inf. Secur. **2**, 218–239 (2004)
32. MacKenzie, P.D., Yang, K.: On simulation-sound trapdoor commitments. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 382–400. Springer, Heidelberg (2004)
33. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: characterizing payments among men with no names. In: Proceedings of the 2013 Internet Measurement Conference. ACM (2013)
34. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. Consulted **1**, 28 (2008)
35. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
36. Paillier Threshold Encryption Toolbox. http://cs.utdallas.edu/dspl/cgi-bin/pailliertoolbox/manual.pdf
37. Pedersen, T.P.: Distributed provers with applications to undeniable signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 221–242. Springer, Heidelberg (1991)
38. Rivest, R., Shamir, A., Adelman, L.: A method for obtaining digital signature and public key cryptosystems. Comm. ACM **21**, 120–126 (1978)
39. Shamir, A.: How to share a secret. Comm. ACM **22**, 612–613 (1979)