Certified Bitcoins

Giuseppe Ateniese[1,2], Antonio Faonio[1],
Bernardo Magri[1], and Breno de Medeiros[3]

[1] Sapienza - University of Rome, Italy
{ateniese,faonio,magri}@di.uniroma1.it
[2] Johns Hopkins University, USA
ateniese@cs.jhu.edu
[3] Google, Inc.
breno@google.com

Abstract. Bitcoin is a peer-to-peer (p2p) electronic cash system that uses a distributed timestamp service to record transactions in a public ledger (called the Blockchain). A critical component of Bitcoin's success is the decentralized nature of its architecture, which does not require or even support the establishment of trusted authorities. Yet the absence of certification creates obstacles to its wider acceptance in e-commerce and official uses. We propose a certification system for Bitcoin that offers: *a*) an opt-in guarantee to send and receive bitcoins only to/ from certified users; *b*) control of creation of bitcoins addresses (certified users) by trusted authorities. Our proposal may encourage the adoption of Bitcoin in different scenarios that require an officially recognized currency, such as tax payments—often an integral part of e-commerce transactions.

1 Introduction

Bitcoin is a peer-to-peer (p2p) electronic cash system, first described in [11]. The Bitcoin p2p network implements a distributed timestamp service that records transactions in a public ledger (called the Blockchain). The timestamp operation is computationally expensive, requiring proof-of-work to verify a transaction and insert it into the Blockchain. In compensation for this effort, the Bitcoin protocol enables the nodes to *mint* coins, i.e., to add into the ledger transactions for self-credit. This distributed minting operation is the source of new currency, dispensing with the need of a central issuer.

Large numbers of users currently transact in Bitcoin, engaging in significantly-sized transactions [13]. The decentralized nature of Bitcoin, wherein confidence on the integrity of the public ledger arises by the cooperative nature of interactions between the participants, is a critical component of its success: Bitcoin removes the necessity for all involved to agree to trust any single entity. However, the converse is also true: Bitcoin does not offer a built-in mechanism to incorporate trustworthiness from real-world entities into the system.

Anonymity In the Bitcoin Protocol. In the Bitcoin Blockchain, users are identified only by addresses, which are pseudonymous public key fingerprints. It is

I. Boureanu, P. Owesarski, and S. Vaudenay (Eds.): ACNS 2014, LNCS 8479, pp. 80–96, 2014.

possible for the user controlling a Bitcoin address to remain unidentified—until information is voluntarily revealed during a purchase or in other circumstances. For this reason Bitcoin has been at times chosen as a payment medium for illegal business. Some governments[1] are also concerned that Bitcoins could be used to skirt capital control laws. On the other hand, legitimate users desirous of privacy should be mindful of the fact that it is possible to link entities that share cash streams—see Ober et al. [13] and Meiklejohn et al. [8] for how an analysis of the Blockchain may reveal that the same real-world entity is behind multiple Bitcoin addresses. Thus, such users should completely segregate their Bitcoin addresses among their different personas.

Our Contribution: Certifiable Bitcoin Addresses. This paper describes an extension of the Bitcoin protocol that preserves its decentralized nature, while also enabling payers to **optionally** specify the involvement of a trusted authority that attests to the identity of the payee, by requiring payees to use *certified* Bitcoin addresses. Conversely, we also enable payees to require that a payer uses a certified Bitcoin address. More specifically, we introduce the concept of Bitcoin addresses that need to be generated with the support of a trusted authority. Those addresses are still anonymous within the Bitcoin system, but the authority can validate the legitimacy of the entity to whom it releases a certified address[2], and other members of the Bitcoin network can attest to the involvement of the trusted authority in issuing the address. These certified addresses are allowed to **co-exist** with the standard auto-generated Bitcoin addresses.

Certified Bitcoin addresses are *blinded*: While the trusted authority can mint coins on behalf of a particular user, it cannot spend any of them. Certified addresses mitigate existing reservations against the adoption of Bitcoin as a currency in commercial uses and against acceptance of the Bitcoin payment protocol as a fully valid alternative to credit card systems.

Identity Theft Mitigation. Our proposal also enhances security against identity theft in Bitcoin. Indeed, consider the case where a man-in-the-middle (MITM) attacker changes the payee's bitcoin address for the attacker's address. For instance, the attacker could deface the payee's website to receive payments intended for the payee. This attack is quite devastating since, in the Bitcoin protocol, once the payment is accepted and registered in the ledger, it is impossible to revert it (unlike credit card payments). With our proposed solution, the payer can first check that the address is certified thus ensuring that the actual identity of the attacker could be recovered by the trusted authority in case of dispute.

1.1 Outline

We briefly recall Bitcoin's transaction mechanism. A Bitcoin transaction is a cryptographically signed statement that transfers an amount of bitcoins from the

[1] China [18] has recently declared Bitcoin illegal.

[2] Note that users may be allowed to use simply, e.g., an email address to request Bitcoin addresses. In this case, an email address, rather than an actual identity, is bound to a Bitcoin address.

sender's to the receiver's address. The sender proves ownership of the bitcoins by "redeeming" a transaction already in the ledger that moves at least the same amount of bitcoins to their address. For more details please refer to Section 3.

The standard approach to add certified addresses to the Bitcoin system would be to use PKI-rooted certificates. A trusted authority would sign each newly released certified address by generating an address certificate. This mechanism can be adapted into Bitcoin's infrastructure by using the Bitcoin scripting language. For a certified address one needs to include a certificate from the central authority to each transaction: A new transaction redeems the earlier one only if *a*) it is verifiable using the sender' address, as with all transactions; and *b*) the attached certificate is valid. However, there are some disadvantages to incorporating a traditional PKI approach in Bitcoin, to wit:

1. A noticeable modification on the software is needed. We need a signature verification operation that takes as input the certified public key (the message), the certificate (the CA's signature on the message) and the public key of the CA in order to verify the certificate. However the operation `OP_CHECKSIG`, which in the Bitcoin scripting language provides signature verification, takes only two inputs – a public key and a signature – and assumes as message the transaction's data. The semantic of `OP_CHECKSIG` would need to be significantly modified or a new operation would have to be added to the scripting language. Any modification of this type would require all the nodes in the system to upgrade their software.
2. In Bitcoin, transaction fees are accounted per bytes: the bigger the size of a transaction, the higher the fees to pay. PKI's addition of certificate chains to (potentially) each address in each transaction would significantly increase transaction costs.
3. The Bitcoin *wallet* software must download the entire ledger. Even an increase of a few gigabytes creates scalability issues, particularly for smartphones or devices with limited bandwidth and data capability. The average size for a block is 156KB and the average number of transactions for each block is 315, which means that the average size of a transaction is approximately 507 Bytes. Considering that the size of a signature in the Bitcoin system's encoding is 71 Bytes, the transaction size will increase by at least 14% [3]. Currently, the size of the ledger is approximately 12GB. In the worst case scenario, where every transaction is being certified, the ledger would be about 1.67 GB bigger.

It would be preferable to add certified addresses in the Bitcoin system *without increasing the size of the transactions* (and, ultimately, the size of the ledger). We achieve this by leveraging the storage and bandwidth cost benefits provided by self-certified public keys. In particular, we adapt techniques developed for self-certified PKI to work within the Bitcoin system. Compared with a standard PKI approach, our solution does not have the drawbacks (2) and (3) outlined above. Moreover, even though we still need to update the software of every

[3] By taking the average over all transactions made in 2013.

node in the network, the modification to accommodate self-signed certificates is easier to accomplish. It can be achieved without changes to the the Bitcoin scripting language, or (in alternative implementation) with minimal changes. Indeed, our solution is perfectly compatible with the current ledger and both systems (standard and certified Bitcoin) can run contemporarily on the same ledger.

1.2 Previous Work

Previous Work on Bitcoin. As pointed out earlier, the Blockchain allows to link entities that share cash streams; and the misconception that pseudonymity provides anonymity has been partially unmasked by a series of recent works on the Bitcoin transaction's graph, see for example [16,13,1]. Previous research has thus focused on *strengthening* the privacy guarantees afforded by Bitcoin. In [4] Barber et al. provide a protocol that features *secure* mixing of money, ensuring money is transferred to fresh, and thus unlinkable, addresses through an untrusted third party. A more radical solution to anonymity is given in the paper of Miers et al. [9], where the authors propose an innovative and Bitcoin-compatible system where full anonymity is achieved via zero-knowledge techniques. In this paper, we focus instead on enhancing trust, via certified bitcoins. Without additional measures, this approach would lower the degree of anonymity in the system; but we point out that our solution is compatible with the approaches proposed in [4,9], allowing for both anonymity and certification within the system.

Other works have focused on improving the scalability of Bitcoin, particularly in what regards the bandwidth required to validate the Blockchain. In [4], the authors proposed a secure filtering service that is backward compatible with the current system. The filtering service sends only relevant transactions to nodes allowing for significant space savings. The service does not increase the degree of linkability, and thus has no impact on the privacy of Bitcoin usage, but the need of a fully-trusted third party can be a deterrent in the Bitcoin's context. Indeed, the filtering service could maliciously hide from the user important transactions—the user needs to fully trust the service provider for the filtering service. In contrast, the trusted authority in our certification scheme is only *functionally trusted*, and a pure enhancement to the Bitcoin's ecosystem. As we shall see in Section 3, the trusted party cannot recover the user secret key. In addition, any abuse from the certification authority are detectable via inspection of the Blockchain.

Another line of research has been recently proposed by Andrychowicz et al. [2] where a general protocol for secure multiparty computation using Bitcoin's transactions is proposed. The system guarantees a form of fairness: if a party interrupts the protocol, the outcome is still "tolerable" to the other honest parties.

Previous Work on Self-Certified Public Key. Our proposal can be seen as a weak version of what is referred to as Self-certified (SC) PKI. SC-PKI contemplates public keys that do not need to be accompanied by a certificate in order to be authenticated by other users. To the best of our knowledge, the first schemes

to rely on only functionally trusted authorities were described by M. Girault in [7]—where the concept of SC-PKI is itself introduced. That work establishes two SC-PKI constructions, one based on RSA and one based on Elgamal-type public keys. It has been later shown that RSA constructions suffer from a drawback, namely it is possible for the trusted party to safely generate its keys to include trapdoor information that facilitates the recovery of other parties' secrets [17]. This attack applies to every RSA-based construction that results in users reconstructing discrete-log type public keys. Therefore, we concentrate on the case where the trusted party's public keys are themselves of discrete-log type.

We note that Girault's SC-PKI schemes are not ideally suited to the desired Bitcoin application. The key generation protocol for the Girault's scheme takes as common inputs the group's parameters $(\mathbb{G}, g)$, the user's identity I and returns as the user's public key the tuple (r, r^s) where $r \in \mathbb{G}$ and the user's secret key is $s \in \mathbb{Z}_q$.

By necessity, the discrete logarithm of r to base g should not be learned by the user, for this would leak the trusted authority's private key. As a result, two public key pairs $(r_A, r_A^{s_A})$ and $(r_B, r_B^{s_B})$ of users A and B are computed with respect to different bases r_A and r_B, respectively. However, this type of public key (i.e., an element in $\mathbb{G}^2$) does not match the Bitcoin specification.

Another self-certified public key scheme based on Elgamal signatures and provably secure in the Random Oracle Model (ROM) was described by Petersen and Horster [14]. The security analysis relies on Pointcheval and Stern's splitting-lemma security arguments [15], and thus achieves only a *loose* reduction to the Discrete Logarithm Problem (DLP).

However, in the Bitcoin setting, a *tight* proof in the Generic Group Model (GGM) is more desirable than a loose proof in the Random Oracle Model. Indeed, the Bitcoin protocol already relies on the security of the ECDSA standard—which is only shown secure via a GGM (tight) reduction to the Elliptic Curve Discrete Logarithm Problem (ECDLP). Our Certified Addresses construction is thus a better fit for Bitcoin in that it is similarly provably secure in the GGM by a tight reduction to the ECDLP—allowing for the entire security analysis to occur within the same well-defined model.

Ateniese and de Medeiros [3] describe a new self-certified scheme based on the Nyberg-Rueppel signature [12] scheme and its variants. The certification scheme in Section 3 can be seen as a novel self-certified scheme where the certification results from the trusted party applying the modified Nyberg-Rueppel signature [3] to the message $m = 0$.

Description of contents. On Section 2 we begin by giving a description of Bitcoin and its transaction mechanism and then we introduce a few standard cryptographic concepts and terminology that will be used in later sections. On Section 3 we present our contribution with a brief description of an implementation. On Section 4 we provide the security analysis of our proposal. Lastly, on Section 5 we give a brief conclusion of the paper.

2 Background

In this section, we provide an overview of the Bitcoin system and its transaction mechanism. We also introduce a few standard cryptographic concepts and terminology that will be used in later sections.

2.1 Bitcoin Signature Scheme

Bitcoin employs the Elliptic Curve Digital Signing Algorithm (ECDSA) [19] for all of its signatures. ECDSA is a widely used and trusted standard, and it has been extensively analyzed. While a security proof for ECDSA in the *Standard Model* is not known, it has been proved secure against existential forgery by adaptive chosen-message attack in the *GGM* [5].

2.2 Bitcoin Transactions

In order to generate a new Bitcoin address (the core identifier in the Bitcoin protocol), a user first produces a pair of private and public keys for ECDSA: (sk, pk). The Bitcoin address relative to (sk, pk) is the hash of the public key, namely $H(pk)$, where H is a hash function based on SHA-256 and RIPEMD-160. Some extra bytes are appended as a checksum.

The simplest case is a *standard* transaction, say with label T_n, between a sender's address S, with public key pk_S, and one recipient address R. The *payload* of this transaction, which we denote by $[T_n]$ contains: an input index p (which refers to the earlier transaction T_p, already committed to the public ledger), the amount v_n of bitcoins to be transferred, the sender's public key pk_S, and the receiver's address R. In addition to its payload, the transaction includes the sender's signature τ on the transaction payload. More precisely, in addition to the payload, the transaction includes a small, standard program in the *Bitcoin Scripting Language* that when executed validates the sender's signature on the payload, by applying the following simple rules: The signature on T_n is *valid* if and only if $H(pk_S) = S$ and the application of the ECDSA verification algorithm with public key pk_S, message $[T_n]$, and signature τ succeeds. That alone is insufficient for T_n to be accepted: In addition, the value v_n being transferred should not exceed the value v_p in the output of the earlier transaction T_p. If this latter condition holds, then T_n can be accepted to *redeem* transaction T_p, provided that the transaction T_p has not been redeemed earlier (otherwise this is an attempt to double-spend the same set of bitcoins, and the transaction should be rejected).

More advance standard transactions with several inputs and several recipient address can be defined. Since such transactions are not necessary for the understanding of this paper we skip their description and refer to [2]. Bitcoin allows the users to also create non-standard (also called *strange*) transactions. Strange transactions have a *validity policy*, specifically, a strange transaction T_p contains in its output a piece of code in the Bitcoin Scripting Language which implements a redemption *policy*. Subsequent strange transaction T_n that purports to redeem

T_p should thus supply any necessary inputs for the evaluation of T_p's policy code, and the transaction T_n successfully redeems T_p if the evaluation of T_p's policy with inputs provided by T_n outputs true (and again under the restriction that no earlier transaction had redeemed T_p).

3 Certified Bitcoin Address

3.1 Description of the Scheme

In this section we describe our main contribution. First we introduce some mathematical concepts and notation. The additive group of integer residues modulo q is denoted by $\mathbb{Z}_q$. The Certification Authority (CA), denoted by T, has the following public parameters: the description[4] $\mathbb{G}$ of a finite group of size q, a generator g of $\mathbb{G}$, and an additional element y_T of $\mathbb{G}$. The private parameter of the CA is the value $x_\mathsf{T} \in \mathbb{Z}_q$ such that $y_\mathsf{T} = g^{x_\mathsf{T}}$. We also fix a function ρ from $\mathbb{G}$ to $\mathbb{Z}_q$. This function could be fairly simple, e.g., it interprets the binary encoding of an element of $\mathbb{G}$ as the encoding of a positive integer.

(Certified Address). A user U can request a certified address to the certification authority T by jointly executing the protocol Certified Key Generation in Table 1. Notice that U samples k uniformly at random in $\mathbb{Z}_q$ (and so does T for k'). At this point, U computes the secret key x and verifies that

$$g^x = c \cdot y_\mathsf{T}^{\rho(c)}.$$

The certified address A is the value $H(c)$.

(Signature Verification). Given a self-certified public key $c \in \mathbb{G}$, the signature verification process works by first extracting the embedded public key y and then using the standard verification. The only operation that needs to be added is the extracting procedure (step 2 on the right of Table 1).

(Certified Transaction). Let S be an address and R a certified address. Before sending bitcoins to the address R, the payer S checks whether there already exists a transaction redeemed by R in the ledger. Notice that R can ensure that such a transaction exists by sending some bitcoins to itself (i.e. a self-transaction). We call the first redeemed transaction of a certified address the *address certification transaction*.

The correctness of the public key derivation follows:

$$g^x = g^{k+k'+\rho(c)x_\mathsf{T}} = g^{k+k'} \cdot g^{\rho(c)x_\mathsf{T}} = c \cdot y_\mathsf{T}^{\rho(c)} \tag{1}$$

For a comprehensive list of the possible interactions between standard and certified addresses we refer to Figure 1.

[4] By description here we mean a (binary) encoding of $\mathbb{G}$ and its operations that can be programmed into a computer.

Table 1. Comparison between Bitcoin and Certified Bitcoin. In both systems, the value A represents the Bitcoin address. The certified key generation is blinded while the transaction verification needs only a single extra step (step 2 on the right). All operations on the exponents are taken modulo q.

Standard Bitcoin	**Certified Bitcoin**
Common inputs: $\mathbb{G}, g$	**Common inputs:** $\mathbb{G}, g, y_{\mathsf{T}}$
Standard Key Generation:	Certified Key Generation:
 **1.** $x \leftarrow \mathbb{Z}_q$ **2.** $y := g^x$ **3.** $A := H(y)$	  **1.** $k \leftarrow \mathbb{Z}_q$ **2.** $h := g^k$ $\xrightarrow{h}$ **3.** $k' \leftarrow \mathbb{Z}_q$ **4.** $c := h \cdot g^{k'}$ **5.** $e := \rho(c)$ **6.** $\bar{x} := k' + e \cdot x_{\mathsf{T}}$ $\xleftarrow[c, \bar{x}]{}$ **7.** $x := \bar{x} + k$ **8.** $A := H(c)$.
Standard Verification:	Certified Verification:
1. **Check** $A = H(y)$; 2. **Check** $\mathsf{Vrf}_y^{ECDSA}([T], \tau)$	1. **Check** $A = H(c)$; 2. **Set** $\boldsymbol{y := c \cdot y_{\mathsf{T}}^{\rho(c)}}$ 3. **Check** $\mathsf{Vrf}_y^{ECDSA}([T], \tau)$

3.2 Implementation Designs

Because of how Bitcoin handles transactions internally, it is not possible to check that an address is certified by just looking at the transaction script. A standard transaction script is shown below:

```
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
scriptSig:    <sig> <pubKey>
```

where `scriptPubKey` is the input script and `scriptSig` is the output script. To verify a transaction, the following actions are performed: (1) after stacking up the signature of the transaction and the redeemer's public key, (2) the latter is hashed by the `OP_HASH160` operation and (3) the hashed valued is compared with the `<pubKeyHash>` value. The problem is that such a value is a hash of the public key and *not* a bitcoin address. The operation `OP_CHECKSIG` is able to

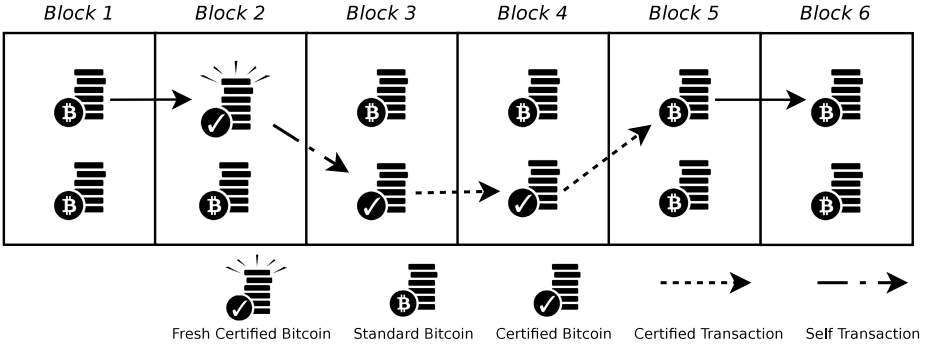

Fig. 1. Certified Bitcoin transactions: The figure shows all possible types of transactions in a ledger with both standard and certified bitcoins. In the second block, a bitcoin is sent to a newly created and supposedly-certified address. This first self transaction in the third block designate that address as indeed certified. In the 5th block, bitcoins are sent from a certified address to a standard address. The last transaction is between standard bitcoin addresses.

distinguish whether the address is certified (by applying the certified signature verification algorithm), but it has no way to report the type of address to the Bitcoin client.

There are a few ways to implement our proposal into the Bitcoin client. We briefly describe three viable options next.

New operation `OP_EXTCERTKEY`. For this implementation, we extend the scripting language with a new operation `OP_EXTCERTKEY` that takes a self-certified public key c as input and then extracts the public key y from it, pushing the extracted key y into the stack, and then re-using the standard signature operation `OP_CHECKSIG` to verify the signature against the extracted key (now in the stack). The size of the transaction would increase by the size of the new operation code (1 byte):

```
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_EXTCERTKEY
              OP_CHECKSIG
```

New operation `OP_CHECKCERTSIG`. Instead, we could extend the scripting language with a new operation `OP_CHECKCERTSIG` that will exclusively handle certified transactions by first extracting the public key y from the self-certified public key c to later perform the standard signature verification. The transaction script for a *certified* transaction replaces the standard operation `OP_CHECKSIG` with this new operation:

```
scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY
              OP_CHECKCERTSIG
```

Modify the operation `OP_CHECKSIG`. Another way is to just modify the client to interpret each transaction as possibly a certified transaction. In this case, the client would first execute the script normally, but if it failed, it would re-attempt the execution using the certified transaction algorithm (i.e., performing an operation such as `OP_CHECKCERTSIG` instead of `OP_CHECKSIG`). The client would then report one of (a) successful standard transaction; (b) successful certified transaction; or (c) verification failure accordingly. The Bitcoin scripting language is unmodified in this approach.

3.3 Security Requirements and Goals

A standard Bitcoin address is self-generated, while a certified address is jointly computed with the involvement of the CA. Thus, it is natural to require that Bitcoin transactions be hard to forge even by a malicious CA. Another security requirement is that certificates must be *unforgeable*—if the adversary does not know the CA's secret key, it cannot generate a certificate and a transaction (signature) through that certificate. Certified Bitcoin addresses share some ideas with both Self-Certified Public Keys [7] and Blind Signatures [6]. The security of our construction holds in the GGM which is the same on which the security of standard Bitcoin relies on. This allows us to provide a security analysis of the protocol within a single and well-defined model.

Crucial to our security formulation is the stipulation that an address be recognized as certified only after it issues a signature (i.e., there is a certified transaction in the public ledger which redeems from it). Indeed, in the absence of the burden of demonstrating knowledge of the secret key, it is trivial for an adversary to "pretend" to have a certificate, since the output c of an (honest-party) execution of the certification protocol is simply an uniformly distributed element in $\mathbb{G}$.

While, by unforgeability of ECDSA, the adversary can not redeem bitcoins from the related address, they may still pretend that the address has been certified. This attack makes no sense in the context of standard Bitcoin addresses: A rational adversary willing to maximize their gain would prefer to exhibit an address for which the secret key is known (to be able to spend any received coins). In our context, on the other hand, if a malicious user falsely claims that an address is certified, it may induce other users to complete unpremeditated transactions.

4 Security of the Certified Bitcoin Addresses

In this section we provide a formal security experiment that captures the informal requirements given in Section 3. Then we show that no PPT generic adversary can win the experiment, providing a formal proof of security to our construction.

We recall that the GGM captures algorithms that access group operations (and indeed the group encoding) through black box function calls. The proofs are inspired by the techniques described in Naccache et al. [10].

In the GGM, the group encoding $\sigma(\cdot) : \mathbb{Z}_q \to \mathbb{G}$ represents an *encoding oracle* that implements a homomorphism from $\mathbb{Z}_q$ onto $\mathbb{G}$.

As before, we employ a function $\rho(\cdot)$ from $\mathbb{G}$ to $\mathbb{Z}_q$, and via notation overload also see $\rho(\cdot)$ as a function from $\mathbb{G}$ to $\mathbb{Z}_q$. To recall, since $\sigma(\cdot)$ encodes elements of $\mathbb{G}$ into binary strings, these strings may thus be interpreted as the binary expansion of an non-negative integer, and that integer can further be *reduced* by computation of its positive remainder modulo q.

In this setting, one describes the public key $y = g^x$ as $\{\sigma(1), \sigma(x)\}$. This notation just means that the homomorphism $\sigma(\cdot)$ maps 1 to g and therefore maps x to y. $\sigma(\cdot)$ is an exponential notation, so x is unrecoverable from $\sigma(x)$.

The group operation oracle $\cdot \oplus \cdot$ takes two encoded group elements $\sigma(v_1)$, $\sigma(v_2)$, and returns the encoded product $\sigma(v_1 + v_2)$. (Since this is exponential notation, the product translate as a sum in the exponents.) Similarly, given $\sigma(v)$ and an integer u, one can implement the square-and-multiply algorithm for exponentiation, using multiple calls to the group operation oracle, to obtain $\sigma(uv)$. One also needs a group inversion oracle $\ominus\sigma(v) \to \sigma(-v)$.

4.1 Unforgeability Formalizations and Proofs

We now provide a rigorous definition of security for the construction in 3.1 by defining the *Signature Unforgeability Experiment*. This is an adversarial game wherein an attacker may obtain one or more certified addresses by executing the protocol with the CA and/or compromise the CA. To succeed in the experiment, the attacker needs to produce a valid message-signature tuple for a fresh certified public key (i.e., one requested by an honest party to the potentially malicious CA).

Notation: k denotes a security parameter, and $\mathsf{poly}(k)$ a value allowed to grow as a polynomial function of k. $\mathcal{A}$ stands for the attacker or adversary. $\mathsf{CKG}_{P,\mathsf{T}}$ represents the Certified Key Generation protocol described in 3.1, where P is some party. In the adversarial game, the adversary has oracle access $\mathcal{O}_\mathsf{T}$ to the trusted party and can execute the protocol $\mathsf{CKG}_{\mathcal{A},\mathcal{O}_\mathsf{T}}$, obtaining new certified keys at will. It may also compromise the trusted party directly, in which case it can execute the protocol $\mathsf{CKG}_{\mathcal{A},\mathsf{T}}$ entirely as a procedure.

The adversary may also request that new (honest) parties P be instantiated and obtain oracle access $\mathcal{O}_P$ with which to execute $\mathsf{CKG}_{\mathcal{O}_P,\mathcal{O}_\mathsf{T}}$ to produce a certified address c for P. Alternatively, if $\mathcal{A}$ has compromised T, it can execute $\mathsf{CKG}_{\mathcal{O}_P,\mathsf{T}}$, which additionally gives it T's view of P's certificate key generation. Finally the adversary can use oracle access $\mathcal{O}_P$ to request signatures $\mathsf{Sign}^{\mathsf{ECDSA}}_{\mathcal{O}_P}(\cdot)$ on arbitrarily chosen messages. The security experiment is described in Fig. 2.

Informally, we say that an adversary wins an experiment if only if the output of the experiment is 1. The security claim is that, under the GGM, there is no efficient adversary that wins the *Signature Unforgeability Experiment*.

Before stating our first security result, we informally recall the hypothesis of the *Security Theorem* of the ECDSA signature scheme in the GGM (Thrm. 2 in D. Brown [5]). The theorem holds under the assumptions that the private keys and ephemeral keys are uniformly random, the hash function is collision resistance and satisfies two other properties called (1) *zero-resistance*, i.e., an

$\mathsf{Exp}^{sig-unf}_{\mathcal{A}}(k)$:

1. $(\mathbb{G}, g, x_{\mathsf{T}}, y_{\mathsf{T}}) \leftarrow \mathsf{Gen}(1^k)$ where $|\mathbb{G}| = \mathsf{poly}(k)$, and set $L \leftarrow \emptyset$, $S \leftarrow \emptyset$;
2. $\mathcal{A}$ with input $\mathbb{G}, g, y_{\mathsf{T}}$ has oracle access to $\mathsf{T} = \mathsf{T}(x_{\mathsf{T}})$ with which can play the protocol $\mathsf{CKG}_{\cdot,\mathcal{O}_{\mathsf{T}}}$
   Let $(c, \bar{x})$ be the output of T after any execution of $\mathsf{CKG}_{\mathcal{A},\mathcal{O}_{\mathsf{T}}}$
   L maintains all certificates whose secrets were produced by $\mathcal{A}$:
   Update $L \leftarrow L \cup \{c\}$;
3. Optionally $\mathcal{A}$ can extract the secret key x_{T} of the trusted party by compromising it;
4. $\mathcal{A}$ may request that arbitrary honest parties P be instantiated. Specifically, an oracle machine $\mathcal{O}_P$ is instantiated and set in pause state;
5. Through oracle access $\mathcal{O}_P$, $\mathcal{A}$ may request that $\mathsf{CKG}_{\mathcal{O}_P,\mathcal{O}_{\mathsf{T}}}$ be executed to output a certified address $c = c_P$ for P ($\mathcal{A}$ has bystander view of an honest party enrollment);
6. If $\mathcal{A}$ has compromised T, it may request that $\mathsf{CKG}_{\mathcal{O}_P,\mathsf{T}}$ be executed, giving $\mathcal{A}$ the trusted party view of an honest party enrollment;
7. $\mathcal{A}$ may request that honest party P sign arbitrary messages m of $\mathcal{A}$'s choice, executing $\tau \leftarrow \mathsf{Sign}^{\mathsf{ECDSA}}_{\mathcal{O}_P}(m) = \mathsf{SignECDSA}_{c_P \cdot y_{\mathsf{T}}^{\rho(c_P)}}(m)$
   Let (c_P, m, τ) be the output after any such execution
   S maintains the set of signatures directly given to A:
   Update $S \leftarrow S \cup \{(c_P, m, \tau)\}$;
8. Eventually $\mathcal{A}$ outputs a triple $s = (c, m, \tau)$; if

$$\mathsf{Vrf}^{\mathsf{ECDSA}}_{y}(m, \tau) = 1 \text{ and } c' \notin L, \text{ and } s \notin S$$

   holds where $y = c \cdot y_{\mathsf{T}}^{\rho(c)}$, then output 1 else 0.

Fig. 2. The $\mathsf{Exp}^{sig-unf}$ experiment

adversary cannot find a message that the hash function maps to 0^k), and (2) *uniformity*, roughly, the distribution of the output value of the hash function on input a uniformly and random message is statistically close to the uniform distribution (see [5] for more details). These two properties are generally believed to hold true in practice for cryptographic hash functions in current usage, in particular the ones employed in the Bitcoin protocol.

Theorem 1. *If the Bitcoin's hash function is collision resistant, zero resistant and uniform, and the ephemeral keys are uniformly random, then there is no efficient, generic adversary that achieves a non-negligible probability of success in* $\mathsf{Exp}^{sig-unf}$.

We prove the theorem by transforming the above experiment into related ones by reasoning about the adversary view (hybrid argument).

First, we note that we can assume that $\mathcal{A}$ always compromises T right after receiving the public parameters. Indeed, the adversarial goal in the experiment is the same whether T is compromised or not, and $\mathcal{A}$'s view of the experiment is strictly enlarged by directly playing the role of T throughout.

Now we look into more detail on the protocol algorithm CKG within the GGM. Whenever a party chooses some random value r and needs to compute $g^r \in \mathbb{G}$, it actually needs to consult the encoding oracle for $\sigma(r)$. The encoding oracle in a GGM simulation maintains a list of previously asked for inputs i_j it has been given and the values it has returned for them: $\{i_1, \sigma_1 = \sigma(i_1), \ldots, i_n, \sigma_n = \sigma(i_n)\}$. If the new input r matches an earlier i_ℓ, it will return the corresponding σ_ℓ. Otherwise, it generates a completely new random binary string z, newly defines $\sigma(r) := z$ and appends to its list $\{i_1, \sigma_1, \ldots, i_n, \sigma_n, i_{n+1} = r, \sigma_{n+1} = \sigma(r) = z\}$, returning z to the caller.

We now modify the experiment simulation to $\overline{\mathsf{Exp}}_{\mathcal{A}}^{sig-unf}$. The only difference between $\overline{\mathsf{Exp}}_{\mathcal{A}}^{sig-unf}$ and $\mathsf{Exp}_{\mathcal{A}}^{sig-unf}$ is as follows: When an honest party P engages with $\mathcal{A}$ (impersonating T) to obtain a certificate, and $\mathcal{A}$ generates k' and attempts to compute $c = hg^{k'} = \sigma(k + k')$, where k is the randomness computed by P; then if the GGM oracle already has some $i_\ell = k + k'$ in its list of previously encoded group elements, the experiment $\overline{\mathsf{Exp}}_{\mathcal{A}}^{sig-unf}$ terminates early, with $\mathcal{A}$ victorious. Let us call Coll this event. Otherwise, it continues identically as $\mathsf{Exp}_{\mathcal{A}}^{sig-unf}$, i.e., the GGM oracle computes an entirely new random string $c = \sigma(k + k')$ and returns it to $\mathcal{A}$.

We claim that $\mathcal{A}$'s additional probability of success in $\overline{\mathsf{Exp}}_{\mathcal{A}}^{sig-unf}$ versus $\mathsf{Exp}_{\mathcal{A}}^{sig-unf}$ is negligible. For if the adversary were able to compute k' such that $k' = i_\ell - k$ for a previously seen i_ℓ, it would also be able to extract the discrete logarithm $k = i_\ell - k'$ from $h = g^k$, given only h. The claim then obviously follows by the standard security assumption of hardness of the Discrete Logarithm Problem in elliptic curves (ECDLP).

Within $\overline{\mathsf{Exp}}_{\mathcal{A}}^{sig-unf}$ it holds that even when honest party P interacts with a malicious trusted party, the protocol execution guarantees that c, and thus the random contribution $\bar{x} = k' + \rho(c) \cdot x_{\mathsf{T}}$ of T to P's private key $x = \bar{x} + k$ is uniformly and randomly distributed.

Specifically, let u be the number of instantiated honest parties P (i.e., u is the number of execution of the CKG protocol between an honest party and the trusted party) and let ϵ bound the probability that a PPT generic algorithm solves the DLP in $\mathbb{G}$, we have that:

$$\begin{aligned}
\Pr\left[\mathsf{Exp}_{\mathcal{A}}^{sig-unf}(k) = 1\right] &\leq \Pr\left[\overline{\mathsf{Exp}}_{\mathcal{A}}^{sig-unf}(k) = 1\right] \leq \\
&\leq \Pr\left[\overline{\mathsf{Exp}}_{\mathcal{A}}^{sig-unf}(k) = 1 \,|\neg\mathsf{Coll}\right] + \Pr\left[\mathsf{Coll}\right] \leq \\
&\leq \Pr\left[\overline{\mathsf{Exp}}_{\mathcal{A}}^{sig-unf}(k) = 1 \,|\neg\mathsf{Coll}\right] + u \cdot \epsilon
\end{aligned}$$

If $\neg\mathsf{Coll}$ holds the experiment guarantees that the honest parties private keys are uniformly and randomly generated. Notice that the experiment $\overline{\mathsf{Exp}}_{\mathcal{A}}^{sig-unf}$ under the condition $\neg\mathsf{Coll}$ is equivalent to u independent and parallel executions of the existential forgery experiment under the chosen-message attack of the ECDSA. We now can directly invoke the security of ECDSA in GGM. In fact, the private keys are uniformly and random, and by hypothesis, the hash function

is collision resistance, zero-resistance and uniform and the ephemeral keys are uniformly and random.
Specifically, let ϵ_{crh} be the probability under collision resistance attack for the underlying hash function, then:

$$\Pr\left[\mathsf{Exp}_{\mathcal{A}}^{sig-unf}(k)=1\right] \leq u\cdot(\epsilon_{crs}+\mathsf{poly}(k)\cdot\epsilon)$$

where the $\mathsf{poly}(k)$ depends on the running time of the adversary. □

In the above we did *not* prove that the generation of certificates itself was unforgeable—indeed, by disclosing T's private key to $\mathcal{A}$ we made it trivial for $\mathcal{A}$ to generate new certificates. Merely proving that a malicious T cannot bias the selection of private keys by honest parties was sufficient given that the Bitcoin construction requires issuing a signature to complete certificate validation. We now consider the issue of unforgeability of certificates separately.

The property is not strictly necessary to the security of the certified Bitcoin address construction, since without a previously seen signature issued by a Bitcoin address, it cannot be considered certified. However, it provides evidence that our Certified Key Generation mechanism can be used in any cryptographic application, provided that the certificate be accompanied by a proof of knowledge of the certificate's associated private key.

We omit a formal definition of security requirements of self-certified public schemes here for conciseness reasons, and instead refer the reader to [7]. More specifically, we provide an adversarial-game formulation of security for the following claim: When the trusted party is honest, adversaries cannot on their own generate certificates on public keys for which they know the private key.

$\mathsf{Exp}_{\mathcal{A}}^{cert-unf}(k)$:

1. $(\mathbb{G}, g, x_{\mathsf{T}}, y_{\mathsf{T}}) \leftarrow \mathsf{Gen}(1^k)$ where $|\mathbb{G}| = \mathsf{poly}(k)$, and set $L \leftarrow \emptyset$;
2. $\mathcal{A}$ with input $\mathbb{G}, g, y_{\mathsf{T}}$ has oracle access to $\mathsf{T} = \mathsf{T}(x_{\mathsf{T}})$ with which can play the protocol $\mathsf{CKG}_{\cdot,\mathcal{O}_{\mathsf{T}}}$
   Let $(c, \bar{x})$ be the output of T after any execution of $\mathsf{CKG}_{\mathcal{A},\mathcal{O}_{\mathsf{T}}}$
   L maintains all certificates whose secrets were produced by $\mathcal{A}$:
   Update $L \leftarrow L \cup \{c\}$;
3. Eventually $\mathcal{A}$ outputs x, c; if

$$y = g^x = c \cdot y_{\mathsf{T}}^{\rho(c)}, \text{ where } c \notin L$$

   holds then output 1 else 0.

The security claim is that, under the GGM, there is no efficient adversary $\mathcal{A}$ that wins the *Certificate Unforgeability Experiment* $\mathsf{Exp}^{cert-unf}$.

Theorem 2. *There is no efficient, generic adversary that achieves a non-negligible probability of success in* $\mathsf{Exp}^{cert-unf}$.

As a generic algorithm, $\mathcal{A}$ works as follows: It maintains a list of linear polynomials $\{F_i\}$, where $F_i = \alpha_i + \beta_i X$, and the coefficients lie in $\mathbb{Z}_q$. The list is

initiated as $\{F_1 = 1, F_2 = X\}$. The algorithm also maintains a list $\{\sigma_i\}$ of encodings, initiated as $\{\sigma_1 = \sigma(1), \sigma_2 = \sigma(x_\mathsf{T})\}$. At the k-th time the algorithm queries the oracle, it provides the indices i, j and a bit b, and the oracle responds with either $\sigma_k = \sigma_i \oplus \sigma_j$ or $\sigma_i \oplus (\ominus\sigma_j)$, according to the case $b = 0$ or $b = 1$, respectively. The algorithm adds σ_k and $F_k = F_i \pm F_j \mod q$ to each of the respective lists, with the $+$ sign being chosen if $b = 0$. (So it is the same sign as in the definition of σ_k in terms of σ_i and σ_j.) Without loss of generality, we may assume that the F_i are distinct linear polynomials with coefficients in $\mathbb{Z}_q$. If, during the execution of the protocol, it happens that $F_i(x) = F_j(x) \mod q$, with $i \neq j$, it follows that $F = F_i - F_j$ is a non-zero polynomial, with $F(x_\mathsf{T}) = 0 \mod q$. This can allow $\mathcal{A}$ to solve it for x_T, thus extracting the trusted party's secret. If the discrete logarithm is hard in $\mathbb{G}$ this can only happen with negligible probability, and we can rule out the occurrence of such execution sequences from the game simulation (called *unsafe* sequences in GGM terminology).

Consider now an algorithm that produces a tuple (c, x), after u queries to the group operation oracle. Note that in this case, the verification equation implies that $c = \sigma(x - \rho(c) \cdot x_\mathsf{T})$. Let $e = \rho(c)$ and $P = x - e \cdot X$. If P is not in the list of oracle queries performed by the algorithm, augment the list by adding $F_{u+1} = P$ at the end, and increment the number of queries $u \leftarrow u + 1$.

Let F_j be the unique appearance of the polynomial P in the list, without loss of generality. Remind that, from the hardness of DL problem in $\mathbb{G}$, there does not exist a index i such that $F_i(x_\mathsf{T}) \equiv F_j(x_\mathsf{T}) \mod q$. This implies that the group operation oracle may return a random value for σ_j, because F_j represents a query for a new encoding when the encoding oracle is called at step j. The probability that σ_j equals c is therefore, no more than $1/|\mathbb{G}|$, as (almost) all values are now equally likely. In other words, the probability that the adversary will arrive at such an execution sequence is $1/|\mathbb{G}|$ for each oracle query, and thus overall negligible if given only a polynomial number of queries. □

Implication of Certificate Unforgeability to Identity Theft Mitigation. Let us briefly consider the implications of certificate unforgeability for our construction, where the certification authority is functionally trustworthy, and indeed collects proofs of (real-world) identity from the entities it certifies. Now, recall the attack scenario where a man-in-the-middle (MITM) attacker changes the payee's Bitcoin address for the attacker's address. Since (by the result above) an attacker cannot forge certificates, the payee has a recourse to report the fraud and bind it to the identity of the malicious party, with cooperation of the CA. To provide a full proof of security of this fact we would have to provide (verbose, but intuitively straightforward) formalizations of CA functional trust and of identity theft in the context of our Bitcoin construction—for reasons of brevity we refrain from expanding on it here.

5 Conclusion

The decentralized nature of Bitcoin is a critical component of its success. In this paper we describe an optional Bitcoin address certification mechanism that in-

corporates trustworthiness from real-world entities into the system, to mitigate against existing reservations to the adoption of Bitcoin as a legitimate currency. We describe how to implement the scheme with the current Bitcoin ledger, allowing certified and non-certified addresses to be used concurrently. In addition, we provide a proof of security within an adversarial-game security model, under the Generic Group Model of computation.

Acknowledgments. This research was supported in part by the PRIN project TENACE.

References

1. Androulaki, E., Karame, G.O., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating user privacy in bitcoin. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 34–51. Springer, Heidelberg (2013)
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, Ł.: Secure multiparty computations on bitcoin. Cryptology ePrint Archive, Report 2013/784 (2013), `http://eprint.iacr.org/`
3. Ateniese, G., de Medeiros, B.: A provably secure nyberg-rueppel signature variant with applications. Cryptology ePrint Archive, Report 2004/093 (2004), `http://eprint.iacr.org/`
4. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better how to make bitcoin a better currency. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 399–414. Springer, Heidelberg (2012)
5. Brown, D.R.L.: The exact security of ecdsa. Technical report, Advances in Elliptic Curve Cryptography (2000)
6. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO, pp. 199–203. Plenum Press, New York (1982)
7. Girault, M.: Self-certified public keys. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 490–497. Springer, Heidelberg (1991)
8. 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 Conference on Internet Measurement Conference, IMC 2013, pp. 127–140. ACM, New York (2013)
9. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: Proceedings of the 2013 IEEE Symposium on Security and Privacy, SP 2013, pp. 397–411. IEEE Computer Society, Washington, DC (2013)
10. Naccache, D., Pointcheval, D., Stern, J.: Twin signatures: an alternative to the hash-and-sign paradigm. In: Proceedings of the 8th ACM Conference on Computer and Communications Security (ACM CCS), pp. 20–27 (2001)
11. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Consulted 1, 2012 (2008)
12. Nyberg, K., Rueppel, R.: A new signature scheme based on the DSA giving message recovery. In: Proceedings of the First ACM Conference on Computer and Communications Security (ACM CCS 1993), pp. 58–61. ACM Press (1993)
13. Ober, M., Katzenbeisser, S., Hamacher, K.: Structure and anonymity of the bitcoin transaction graph. Future Internet 5(2), 237–250 (2013)

14. Petersen, H., Horster, P.: Self-certified keys – concepts and applications. In: Proceedings of the Third Conference on Communications and Multimedia Security. Chapman & Hall (1997)
15. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)
16. Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 6–24. Springer, Heidelberg (2013)
17. Saeednia, S.: A note on girault's self-certified model. Information Processing Letters 86(6), 323–327 (2003)
18. Wired.com. Bitcoin bubble bursts as china cracks down on digital currency (December 2013), `http://www.wired.com/wiredenterprise/2013/12/china_crackdown/`
19. X9.62-2005, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Standard (ECDSA) (November 2005)