Increasing Anonymity in Bitcoin

Amitabh Saxena[1], Janardan Misra[1], and Aritra Dhar[2]

[1] Accenture Technology Labs, Bangalore 560066, India,
{amitabh.saxena, janardan.misra}@accenture.com
[2] Indraprastha Institute of Information Technology, New Delhi, India
aritra1204@iiitd.ac.in

Abstract. Bitcoin prevents double-spending using the *blockchain*, a public ledger kept with every client. Every single transaction till date is present in this ledger. Due to this, true anonymity is not present in bitcoin. We present a method to enhance anonymity in bitcoin-type cryptocurrencies. In the blockchain, each block holds a list of transactions linking the sending and receiving addresses. In our modified protocol the transactions (and blocks) do not contain any such links. Using this, we obtain a far higher degree of anonymity. Our method uses a new primitive known as *composite signatures*. Our security is based on the hardness of the Computation Diffie-Hellman assumption in bilinear maps.

Keywords: bitcoin, cryptocurrency, aggregate signatures, plausible deniability, anonymity

1 Introduction

Bitcoin (symbol ฿) is virtual currency based on peer-to-peer technology. It is designed to operate without any central authority and enables transaction confirmation using a reward system [1, 2]. The first transaction of every block is a reward (currently ฿25) to whoever first provides a solution to a hard puzzle as a "proof-of-work". The puzzle is constructed from unconfirmed transactions and the proof-of-work serves as a tamper-proof ledger.

In bitcoin, funds are exchanged between *addresses* which are hashes of public keys[3] The addresses serve as pseudonyms and provide some anonymity. However, bitcoin raises serious privacy concerns because all the information is public and permanently stored. Furthermore, digital signatures used in transactions provide cryptographic proofs of funds transfer.

Our contribution: We propose a method to enhance the anonymity of bitcoin-type currencies using a new primitive known as *composite signatures*. Our method removes any cryptographic proofs of funds transfer. The links between inputs and outputs are obfuscated using composite signatures. In this method, multiple transactions are grouped together into a larger transaction to hide the links

[3] We use the terms 'address' and 'public key' interchangeably. The meaning will be clear from the context.

of the individual transactions. Our anonymity comes in the form of *plausible deniability* [3] ("you may have done it but we cannot be sure").

The rest of the paper is organized as follows. We review related works in Section 2. We give an overview of bitcoin in Section 3. We describe our method to enhance anonymity using composite signatures in Section 4. We give the definition and construction of composite signatures in Section 5. A summary of our method is given in Section 6. Finally, we describe how to integrate our protocol with existing bitcoin protocol in Section 7.

2 Related Work

Aggregate signatures: In aggregate signatures [4] many individual signatures can be combined into and replaced with one short object - the aggregate signature. They were proposed to increase efficiency in verifying multiple signatures in applications such as routing protocols.

Composite signatures: The aggregate signatures of [4], however, have another useful property that is not captured (and not needed) in standard definitions such as in [4]. The property is that the aggregation process is *one-way* - given just the aggregate signature, it is very hard to compute the individual signatures. This was used in *verifiably encrypted signatures* [4]. Coron and Naccache proved in [5] that extracting any sub-aggregate signature in a *non-adaptive attack* (where the adversary makes only one sign query) is as hard as solving the CDH problem. Composite signatures capture this property in the stronger *adaptive chosen key and message attack*, where the adversary is allowed to make several sign queries on messages of his choice before outputting a forgery.

There are other extensions of aggregate signatures such as sequential aggregate signatures [6–8], ordered multi-signatures [9, 10], history-free sequential aggregate signatures [11] and sequential aggregate signatures with lazy verification [12]. However, none exploit the one-way property of aggregate signatures.

Anonymity in Bitcoin: Elli Androulaki et.al [13] discuss privacy issues in bitcoin such as discovering which public addresses are controlled by the same user. They classify the problem into Activity Unlinkability or Address Unlinkability and User Profile Indistinguishability. and propose several heuristic techniques to reveal user privacy in multi-input transactions. Furthermore, they perform behavioral analysis to link multiple public addresses to same user. Fergal Reid and Martin Harrigan [14] on the other hand considered the topological structure of two networks derived from bitcoin's public transaction history and analyze implications for the anonymity and currency theft. In [15], Dorit Ron and Adi Shamir used the transaction graph of [14] to find that several large transactions were likely used to obfuscate the funds from a larger transaction earlier on.

Current and proposed approaches for increasing anonymity rely on "mixers" that mix bitcoins from various different sources before sending to destinations. Zerocoin [16] is a technique that uses zero-knowledge proofs and commitment

schemes to unlink sending and receiving addresses and uses an alternate currency as an intermediate exchange medium. Our technique does not rely on alternate currencies or zero-knowledge proofs. Note that although our method also does not provide true anonymity, the anonymity offered is far higher than what is currently offered in bitcoin. Our method can be used in conjunction with other proposed approaches (such as zerocoin). Compared to zerocoin, it is easier to integrate our method with bitcoin.

The CoinJoin [17] protocol is similar to ours. CoinJoin's goal is to unlink inputs in the same wallet. Several parties agree on the inputs and outputs of a transaction. The total funds in the inputs should cover the total funds of the outputs. Finally, the parties individually sign the transaction for the inputs they control. Once all the inputs are signed, the transaction is broadcast. The difference in our method is that the input/outputs of other parties need not be known a priori. Additionally, our method is non-interactive while CoinJoin requires interaction with other parties. Finally, in CoinJoin, parties cannot later deny knowledge of the outputs and other inputs.

3 Overview of Bitcoin

Although the bitcoin protocol is quite complex [18], only a few basic concepts are necessary to understand our idea. These are: *transaction*, *input*, *output*, *reference*, *block* and *confirmation*. We describe them below.

Transaction: Roughly speaking, a transaction consists of a set of *inputs* (source of funds) and *outputs* (destination of funds).

Example: Suppose Alice is the owner of address A which received x bitcoins in a previous transaction. She wants to send $y \leq x$ bitcoins to Bob's address B. Alice constructs a transaction with A as the input and B as one of the outputs. She also inserts ref, the reference to the previous transaction's output where A received those x bitcoins. The entire amount x must be transferred from A. Alice sends y bitcoins to B, sets a transaction fee t and sends the remaining amount $z = x - y - t$ to her *change address* C, which is the other output. The change address is simply any address owned by Alice (possibly A). The message

$$\text{“}(ref\text{: remove } \text{฿}x \text{ from } A), (\text{put } \text{฿}y \text{ in } B), (\text{put } \text{฿}z \text{ in } C)\text{”}$$

is signed under A.

Notation: We will use the following notation:

- $X \stackrel{ref}{\rightarrow} x$ is the message “(ref: remove ฿x from X)”. This is an input.
- $X \leftarrow x$ is the message “put ฿x in X”. This is an output.
- $\sigma_X(m)$ is signature on message m under public key X.

Alice's transaction is then $(m, \sigma_A(m))$, where $m = (A \stackrel{ref}{\rightarrow} x, B \leftarrow y, C \leftarrow z)$.

Transactions: The above scenario had a single input. In reality, a bitcoin transaction can have multiple inputs with no particular link between any source-destination pair. The entire transaction is signed under every input public key. The only requirement is that the sum of the funds at the inputs is greater than or equal to the sum of funds at the outputs. Any difference is considered a transaction fee. More formally, define m to be the message

$$M \stackrel{\text{def}}{=} (A_1 \stackrel{ref_1}{\rightarrow} x_1, A_2 \stackrel{ref_2}{\rightarrow} x_2, \ldots, A_n \stackrel{ref_n}{\rightarrow} x_n, B_1 \leftarrow y_1, B_2 \leftarrow y_2, \ldots, B_l \leftarrow y_l),$$

where: $(A_1, x_1, ref_1), (A_2, x_2, ref_2), \ldots, (A_n, x_n, ref_n)$ are n tuples each consisting of an address A_i, amount of funds x_i and a reference to a previous transaction where A_i received x_i bitcoins, and $(B_1, y_1), (B_2, y_2), \ldots, (B_l, y_l)$ are l pairs of addresses and amount of funds. A valid transaction tx is a tuple:

$$tx \stackrel{\text{def}}{=} (M, \sigma_{A_1}(M), \sigma_{A_2}(M), \ldots, \sigma_{A_n}(M)) \tag{1}$$

such that each signature $\sigma_{A_i}(M)$ verifies correctly and the following holds:

1. $\sum_{i=1}^{l} y_i \leq \sum_{i=1}^{n} x_i$
2. Each ref_i for $1 \leq i \leq n$ was never used in any prior transaction.

The ordering of the signatures in tx is determined from the ordering of messages inside M (which is fixed due to the signatures).

Referencing outputs: In future, when spending the funds from any of the outputs (say $B_i \leftarrow y_i$) of the above transaction, a reference $ref_{B_i \leftarrow y_i}$ to that output needs to be provided. Let tx be the string of Eqn. 1. Then

$$ref_{B_i \leftarrow y_i} \stackrel{\text{def}}{=} (Hash(tx), i)$$

Because ref is constructed from the hash of a previous transaction, it is guaranteed that two different transactions are distinct unless the outputs, input and ref are identical (a forbidden scenario). Due to this, it is also guaranteed (with high probability) that the refs generated by using hashes of two different transactions are also different. In fact, this is how bitcoin prevents double spending (see below). A ref can be used in a transaction at most once. Bitcoin clients maintain a list of unused refs to do this check.

Unspent outputs (and double-spends): An unspent output is essentially an unused reference, one that has never been used in any transaction. The protocol design guarantees that references to two different outputs will be distinct (see above). Each client maintains a set called 'unspent outputs'. Each output of every transaction is added to this set, and removed when is it used as a reference in another transaction. A transaction with a reference not in this list is considered a double spend and is not processed.

Validating Transactions: A new transaction is valid if all the references are unused. If so, the transaction is accepted as *valid* but *unconfirmed*, and is relayed on the network. The clients add each such transaction to a pool of unconfirmed transactions. Unconfirmed transactions can be double-spent.

Confirming Transactions: A miner is a client who confirms new transactions by solving a hard puzzle and providing the solution as a 'proof-of-work' as follows:

1. A bunch of unconfirmed transactions along with one reward transaction (known as the *coinbase transaction*) are combined into a 'block'.
2. Hash of the previous block h_{pr} is added to the block.
3. A nonce is added to the block.
4. Hash(b) of the final block b is computed.

If the output of the hash contains at least a specified number of leading zeros, the puzzle is solved, otherwise the miner tries with different nonces until the puzzle is solved or some other miner broadcasts the solution of a puzzle for a block referencing h_{pr}. A correct solution implies that the corresponding block is 'mined' and all transactions contained in it are confirmed.

Confirmations: The number of confirmations of a transaction are the number of blocks in the blockchain that have been accepted by the network since the block that includes the transaction. The possibility of double-spending a transaction decreases exponentially with the number of confirmations. The default client requires 6 confirmations for normal transactions and 100 confirmations for reward transactions before they can be spent.

Transaction pool management: Each client maintains a pool of unverified (but valid) transactions. An element is removed from this pool when that transaction gets included in a mined block. This ensures that even if a transaction is not included in an immediate block, it is kept in the pool until it gets mined.

Anonymity: Transactions are *not* anonymous; since each input public-key signs the entire transaction, some information is inherently leaked. In particular,

1. Each output is linked to the inputs via the signatures.
2. Each input is also linked to the previous output via the ref.
3. The inputs themselves are linked together (they belong to the same wallet).

4 Increasing Anonymity

The links between inputs and outputs result in loss of anonymity. We describe a slight modification to the protocol that removes these links. The modification is so minor that apart from the way signatures and references are computed, the rest of the design remains the same. Yet, the anonymity gained is significant.

The intuition for anonymity is that because inputs and outputs in a transaction are linked cryptographically, a miner and other intermediaries can 'dilute' the information contained in a transaction by inserting more information before processing it further. The final mined block will have the input-output links in each individual transaction highly obfuscated. The only information will be the set of inputs and outputs of an entire block.

Our protocol uses a primitive called *composite signatures* described below.

Composite signatures. The symbol $\sigma_X(m)$ denotes a signature on message m under public key X. Roughly speaking, composite signatures are an extension of aggregate signatures with the following properties:

1. **Composition:** A number of individual signatures $\sigma_{X_1}(m_1), \sigma_{X_2}(m_2), \ldots, \sigma_{X_n}(m_n)$ can be combined into a composite signature $\sigma_{\{X_1,X_2,\ldots,X_n\}}(\{m_1, m_2, \ldots, m_n\})$, which proves that each m_i was signed under public key X_i.
   The composite signature is said to be on the set $\{(m_1, X_1), (m_2, X_2), \ldots (m_n, X_n)\}$.
2. **Incremental composition:** More signatures can be added to the composite signature at any time.
3. **One-way:** It is computationally hard to obtain any sub-composite signature given just the composite signature. Informally, given the composite signature on a set $S = \{(m_1, X_1), (m_2, X_2), \ldots (m_n, X_n)\}$ of (message, public-key) pairs, it is hard to compute the composite signature on any subset $S' \subsetneq S$.
4. **No ordering:** The signature does not maintain order. It is impossible to decide if a composite signature was computed 'all at once' or incrementally.

Composite signatures are formally defined in Section 5.

A modified protocol Consider the message from the original protocol:

$$M \stackrel{\text{def}}{=} (A_1 \stackrel{ref_1}{\rightarrow} x_1, A_2 \stackrel{ref_2}{\rightarrow} x_2, \ldots, A_n \stackrel{ref_n}{\rightarrow} x_n, B_1 \leftarrow y_1, B_2 \leftarrow y_2, \ldots, B_l \leftarrow y_l),$$

M is a combination of messages $m_1, m_2, \ldots, m_n, \overline{m}_1, \overline{m}_2, \ldots, \overline{m}_l$, where:

$$m_i \stackrel{\text{def}}{=} (A_i \stackrel{ref_i}{\rightarrow} x_i) \quad (1 \leq i \leq n) \qquad \text{[Inputs]}$$

$$\overline{m}_i \stackrel{\text{def}}{=} (B_i \leftarrow y_i) \quad (1 \leq i \leq l) \qquad \text{[Outputs]}$$

Transactions: Instead of defining a transaction as in Eqn. 1 (repeated below):

$$tx \stackrel{\text{def}}{=} (M, \sigma_{A_1}(M), \sigma_{A_2}(M), \ldots, \sigma_{A_n}(M)),$$

we define it using composite signatures as follows::

$$tx \stackrel{\text{def}}{=} (M, \sigma_{\{A_1,A_2,\ldots,A_n,\overline{A}_1,\overline{A}_2,\ldots,\overline{A}_l\}}(\{m_1, m_2, \ldots, m_n, \overline{m}_1, \overline{m}_2, \ldots, \overline{m}_l\})), \quad (2)$$

such that each $\overline{A}_i$ is a randomly generated public key, called a *masking key*, and the pairs $(\overline{A}_i, \overline{m}_i)$ are unique. Define $\Lambda \stackrel{\text{def}}{=} \{A_1, A_2, \ldots A_n, \overline{A}_1, \overline{A}_2, \ldots \overline{A}_l\}$ and $\Pi \stackrel{\text{def}}{=} \{m_1, m_2, \ldots m_n, \overline{m}_1, \overline{m}_2, \ldots \overline{m}_l\}$. Equivalently, $tx \stackrel{\text{def}}{=} (\Pi, \sigma_{(\Lambda)}(\Pi))$.

Observe that in the above transaction, unlike the original bitcoin protocol, each 'regular' public key signs a message containing only its own address. Consequently, the signatures never link the sending addresses to the receiving addresses or other sending addresses. The one-way property of composite signatures preserves the security of the original protocol; it is infeasible to isolate any signatures spending funds from the inputs.

Confirming a transaction: A transaction tx is valid if each of the inputs has an unused reference to a previous output. Confirmation of tx requires a miner to solve a puzzle for a block containing that transaction, constructed as follows:

1. A number of unconfirmed transactions $tx_1, tx_2, \ldots tx_\alpha$ are collected for inclusion in the block, where each tx_i is defined as :

$$tx_i \stackrel{\text{def}}{=} (\Pi_i, \sigma_{\Lambda_i}(\Pi_i)) \qquad (1 \leq i \leq \alpha)$$

   Additionally, a coinbase (reward) transaction tx_c with no inputs is created:

$$tx_c \stackrel{\text{def}}{=} (\Pi_c, \sigma_{\Lambda_c}(\Pi_c)),$$

2. It is verified that each (masking-key, output) pair from all the transactions combined together is unique. Not only do we require that the pairs are unique in each transaction but also in all the transactions combined together.
3. A final block b is computed as follows:
   (a) Hash of the previous block h_{pr} is computed.
   (b) A combined composite signature σ_b is computed. That is,

$$\sigma_b \stackrel{\text{def}}{=} \sigma_{(\Lambda_c \cup \Lambda_1 \cup \Lambda_2 \cup \ldots \Lambda_\alpha)}(\Pi_c \cup \Pi_1 \cup \Pi_2 \cup \ldots \Pi_\alpha)$$

   (c) Assume some canonical ordering of all inputs and outputs. Define

$$\Pi_b \stackrel{\text{def}}{=} \Pi_c \cup \Pi_1 \cup \Pi_2 \cup \ldots \Pi_\alpha,$$

   where the elements of Π_b are arranged in the canonical order.
   (d) The final mined block b is computed as:

$$b \stackrel{\text{def}}{=} (h_{pr}, \Pi_b, \sigma_b, \theta_b),$$

   where θ_b is a nonce s.t. $Hash(b)$ has a certain number of leading zeros.

Referencing the outputs: In this modified protocol, we don't reference simply the outputs, but rather the (masking-key, output) pairs. Let $(\overline{A}_j, \overline{m}_j)$ be some (masking-key, output) pair in one of transactions included in the above block. Recall that such a pair is unique in a block (even if the output may be repeated). We compute a reference to the above pair as:

$$ref_{(\overline{A}_j, \overline{m}_j)} \stackrel{\text{def}}{=} (Hash(b), Hash(\overline{A}_j, \overline{m}_j))$$

Since the reference contains the hash of the block, an output can only be spent if its transaction has been included in a mined block. This makes the new transaction incompatible with services that allow spending from unconfirmed transactions (such as satoshidice.com). However, this also makes the protocol more robust to DoS attacks. To summarize, in the modified protocol, **it is not possible to spend from unconfirmed transactions**.

Security: Composite signatures provide security against two distinct types of forgery. The first type, called *ordinary forgery* is the one that all conventional signature schemes are expected to satisfy. This involves forging a signature under an input public-key to steal funds. The second one, called *extraction forgery* occurs when two signatures can be 'separated' given their composition. This will also allow an attacker to steal funds.[4] Since extraction of any sub-composite signature is infeasible, peers can only add further signatures to a transaction. Double spending and replay attacks are prevented in a manner similar to the original protocol. We maintain a list of unused refs, and reject the transaction that contains a ref that has been used. The references are unique because:

1. The reference is a hash of the block and the (masking-key, output) pair.
2. Each block is unique because it contains a hash of the previous block.
3. The (masking-key, output) pairs in a block are unique.

We additionally consider the case where the sender uses a weak or compromised masking key. This is similar to a double spending attack. The receiver should not trust the transaction until it is confirmed.

Anonymity: First observe that each input and output is cryptographically linked to only one public key (the regular key or a masking key). Therefore given a transaction as in Eqn 2, it is impossible to prove that the signer knew any outputs. Furthermore, signatures from many transactions can be composed to obfuscate the input-output relationships (we discuss this below). Additionally, once a transaction is confirmed in a block, it is removed from memory and only the confirmed block is stored. The block alone does not leak any information about the input-output links. Consequently, if the individual transactions are not saved, this information is eventually erased with time.

Enhancing Anonymity: We can enhance anonymity via *plausible deniability*.

Joiners: To further enhance anonymity, we propose the notion of *joiners* as follows. The senders will leave a certain amount of funds free for their peers (this is additional to the transaction fee). This transaction is called *partial* and the free funds are the *joining bonus*. This transaction is sent to only one peer. Peers receiving any transaction with free funds can add their addresses as outputs and claim the joining bonus to make the transaction *full* before broadcasting it to the network. The joining bonus is not specifically marked to make it indistinguishable from normal funds. Given a full transaction, it should not be possible to distinguish which outputs consume the joining bonus.

Even with access to the individual transactions, it would still be impossible to prove with certainty that the sender indeed sent those funds to some given output, since it is possible that the outputs were added later on by a joiner. To ensure that transaction fees don't get consumed by joiners, a special output

[4] If an attacker can extract signatures, he can isolate the input and add any output.

can be used for transaction fees. To ensure that the original partial transaction is never broadcast, a spender should send it to only to one peer. Once the transaction is full, it will be broadcast to the network. Clients attempting to disrupt the network by broadcasting partial transactions will be handled as explained below in the section on transaction pool management. Similarly if a misbehaving peer drops a partial transaction, this can be detected and the transaction resent to a different peer.

Merging services: A merging service accepts various transactions from clients (over a private channel) and once sufficient of them are obtained, it merges them by aggregating the signatures before broadcasting to the network. Clients attempting to disrupt the network by sending the same transactions to multiple merging services will be handled in a similar way as for joiners.

Using the Knapsack problem: Given a 'merged' or 'joined' transaction, it may still be possible to deduce some input-output relationships from the amount of funds going in and out. We use the knapsack problem to further obfuscate this information. The knapsack problem [19–21] can be described as follows. Given a positive rational number X and a set W of positive rational numbers $w_1, w_2, ...w_n$, find a subset S of W (if it exists) such that $Sum(S) \leq X$.

The recipient generates a number of addresses to receive funds into. The sender randomly splits the funds into those addresses and broadcasts the transaction. Other joiners/merging services add further transactions also generated in a similar manner. In the merged transaction, any subset of inputs and outputs with sums close to each other can potentially belong to one sub-transaction. However, if there are overlapping or multiple solutions, then it is impossible to prove this fact with certainty, thereby ensuring plausible deniability.

Revealing the masking keys: To enhance deniability, the masking private keys can be publicized after a few confirmations. In this case, a partial transaction (presented later) cannot act as a cryptographic proof of knowledge of the outputs.

Transaction pool management: Referring to the joiner protocol above, suppose a malicious peer transmits a partial transaction $tx = A$ to j joiners, where A is a set of inputs and outputs. This will result in multiple full transactions $AB_1, AB_2, \ldots AB_j$, one for each joiner. Since an output can only be used once, only one of these transactions will be accepted. In such a situation, a peer will reject all new transactions, while a miner could pick one that maximizes fees.

The remaining aspects of the protocol such as rules for pruning and broadcasting shall be the same as the original protocol.

5 Composite Signatures

Our protocol uses a primitive called composite signatures, which we define here.

Message-descriptor: A message-descriptor is a set $\{(m_1, pk_1), (m_2, pk_2), \ldots, (m_n, pk_n)\}$ of (message, public-key) pairs.

Algorithms. A composite signature scheme has four algorithms:

1. **KeyGen**(K) The algorithm takes in a security parameter K and outputs a (public-private) key pair pk, sk.
2. **Sign**(sk, m) The algorithm takes in a private key sk and a message m. It outputs a single-key signature σ. This single-key signature is equivalent to a composite signature on the single pair $\{(m, pk)\}$
3. **Compose**$((\ell_1, \sigma_1), (\ell_2, \sigma_2))$ The algorithm takes in two (message-descriptor, signature) pairs. If both signatures are valid and $\ell_1 \cap \ell_2 = \emptyset$, it outputs a composite signature σ on the message-descriptor $\ell_1 \cup \ell_2$, otherwise it outputs an error symbol $\perp$. Validity is checked by the **Verify** algorithm below.
4. **Verify**(ℓ, σ) The algorithm takes in a message descriptor

$$\ell = \{(m_1, pk_1), (m_2, pk_2), \ldots, (m_n, pk_n)\},$$

and σ, a purported composite signature on ℓ. If the messages in ℓ are not unique, the algorithm outputs `Invalid`. Otherwise it invokes a deterministic poly-time procedure and outputs either `Valid` or `Invalid`.

The composite signatures exhibit an abelian group structure under composition. Given a set of composite signatures, we can compute composite signatures on any union of their message descriptors. Furthermore, these are the only composite signatures we should be able to compute. We capture this below.

Security Security is defined using the following interaction with a forger A.

1. *Setup:* A chooses n. We generate n (public-private) keypairs $\{(pk_i, sk_i)\}_{i \in [1..n]}$ with security parameter K. We give the set $PK = \{pk_i\}_{i \in [1..n]}$ to A.
2. *Queries:* A makes up to α *sign* queries. Each sign query i consists of ℓ_i, a message-descriptor with public keys from PK. If the pairs in ℓ_i are unique, we respond with an composite signature on ℓ_i, otherwise we return the error symbol $\perp$. Let L be the set of message-descriptors in all sign queries.
3. *Output:* A outputs (ℓ_A, σ_A), a purported (message-descriptor, signature) pair possibly containing public keys not from PK. Let $PK_A = \{pk | (m, pk) \in \ell_A\}$. A wins if the following conditions hold:
   (a) **Verify**$(\ell_A, \sigma_A) =$ `Valid`.
   (b) The set $PK \cap PK_A$ is non-empty.
   (c) ℓ_A is *not signable* (Def. 1 below).

Notation: Let $\ell'_A = \{(m, pk) | (m, pk) \in \ell_A \wedge pk \in PK\}$. Assign a unique prime number to each element of the set $\{(m, pk) | ((m, pk) \in \ell \wedge \ell \in L) \vee (m, pk) \in \ell'_A\}$. Then each $\ell \in L$ corresponds to a unique integer $integer(\ell)$ obtained by multiplying the primes corresponding to its constituent (m, pk) pairs. Let Z be the set $\{integer(\ell) | \ell \in L\}$. Let $z_A = integer(\ell'_A)$, obtained by multiplying the primes corresponding to ℓ'_A.

Definition 1 *(Signable Set) The set ℓ_A is* ***signable*** *iff there exists a solution in* ***non-negative integers*** *x_i to the equation $z_A = \prod_{z_i \in Z} z_i^{x_i}$.*
In a weaker notion, we allow integer solutions. We call this weakly *signable.*

Example. Suppose $L = \{\ell_1, \ell_2, \ell_3\}$, with $\ell_1 = \{(m_1, pk_1), (m_2, pk_2)\}$, $\ell_2 = \{(m_2, pk_2), (m_3, pk_3)\}$ and $\ell_3 = \{(m_3, pk_3), (m_4, pk_4)\}$. Let $\ell_A = \{(m_1, pk_1), (m_4, pk_4)\}$. Let us assign the primes as: $(m_1, pk_1) \to 2, (m_2, pk_2) \to 3, (m_3, pk_3) \to 5, (m_4, pk_4) \to 7$. We have $Z = \{6, 15, 35\}$ and $z_A = 14$. Then ℓ_A is weakly signable because $14 = 6 \cdot 15^{-1} \cdot 35$. However, ℓ_A is not signable since there are no solutions in non-negative integers to $14 = 6^{x_1} \cdot 15^{x_2} \cdot 35^{x_3}$.

Observe that the signable sets form a monoid under the signature aggregation operation, while the weakly signable sets form a group. The signable sets are exactly those sets that can be generated by aggregating the collected signatures using this operation.

Definition 2 *A composite signature scheme {KeyGen, Sign, Compose, Verify} is secure if for sufficently large K, there is no probabilistic poly-time A that wins with non-negligible advantage in K.*

Intuition: In the above definition, aggregation of signatures is represented by multiplication of the primes. The game captures the fact that it is possible to generate new signatures by aggregating smaller signatures (represented by signable numbers - obtained by multiplying elements of Z). Furthermore, it may additionally be possible to generate new signatures by 'reversing the aggregation algorithm' when only one input is unknown (represented by weakly signable numbers - obtained by multiplying *and dividing* elements of Z).

Construction. Our construction is derived from the aggregate signatures of [4] by appending the public key and a random string to the message.

Bilinear pairing: Let G_1 and G_2 be two cyclic multiplicative groups both of prime order q. A bilinear pairing is a map $\hat{e} : G_1 \times G_1 \mapsto G_2$ satisfying:

- *Bilinearity*: $\hat{e}(a^x, b^y) = \hat{e}(a, b)^{xy}$ $\forall a, b \in G_1$ and $x, y \in \mathbb{Z}_q$.
- *Non-degeneracy*: If g is a generator of G_1 then $\hat{e}(g, g)$ is a generator of G_2.
- *Computability*: The map $\hat{e}$ is efficiently computable.

We require a case where the discrete logarithm problem in G_1 is believed to be hard. Such bilinear pairings are known to exist (see [4]). Our security depends on the hardness of the following problem in G_1:

Computation Diffie-Hellman (CDH) problem: Given g^x, g^y for a generator g of G_1 and unknowns $x, y \in \mathbb{Z}_q$, compute g^{xy}.

Algorithms: Select a security parameter κ. Let $\hat{e} : G_1 \times G_1 \mapsto G_2$ be a bilinear map over groups (G_1, G_2) of prime order q, and g be a generator of G_1. Denote by Σ the alphabet $\{0, 1\}$. Let $H : \Sigma^* \times \Sigma^\kappa \times G_1 \mapsto G_1$ be a cryptographic hash function. These parameters are public.

1. **KeyGen:** The private key is $x \xleftarrow{R} \mathbb{Z}_q$ and the public key is $pk = g^x \in G_1$.
2. **Sign:** To sign a message m under the above public key pk, generate $r \xleftarrow{R} \Sigma^\kappa$ and compute the signature $\sigma \in (G_1, \Sigma^\kappa)$ as:

$$\sigma = (H(m, r, pk)^x, r)$$

3. **Compose:** Two (message-descriptor, signature) pairs, $(\ell_1, \sigma_1), (\ell_2, \sigma_2)$ are given. Ensure that $\textbf{Verify}(\ell_1\sigma_1) = \textbf{Verify}(\ell_2, \sigma_2) = \texttt{valid}$ and $\ell_1 \cap \ell_2 = \emptyset$. Then parse σ_1 and σ_2 as (σ_1', R_1) and (σ_2', R_2) respectively and compute the composite signature σ on $\ell_1 \cup \ell_2$ as $\sigma = (\sigma_1'\sigma_2', R_1 \cup R_2)$.
4. **Verify(ℓ, σ):** Here $\ell = \{(m_1, pk_1), (m_2, pk_2), \ldots, (m_k, pk_k)\}$ is a message-descriptor of length k and σ is a purported composite signature on ℓ. To verify σ, first ensure that all pairs are distinct. Then parse σ as $(\sigma', \{r_1, r_2, \ldots, r_k\}) \in G_1 \times (\Sigma^\kappa)^k$ and check that the following holds:

$$\hat{e}(\sigma', g) \stackrel{?}{=} \prod_{i=1}^{k} \hat{e}(H(m_i, r_i, pk_i), pk_i)$$

Verification works because:

$$LHS = \hat{e}(\prod_{i=1}^{k} \sigma_i', g) = \hat{e}(\prod_{i=1}^{n} H(m_i, r_i, pk_i)^{x_i}, g) = \prod_{i=1}^{n} \hat{e}(H(m_i, r_i, pk_i), g^{x_i}) = RHS$$

Security: Security is based on the hardness of the CDH problem (Theorem 1).

Theorem 1. *Let H be a random oracle and let ϵ be the probability of an attacker breaking the composite signature scheme after making at most α sign queries and at most γ queries to H, such that the forgery contains at most β keys. Then we can solve the CDH problem in G_1 with probability $\geq \frac{\epsilon}{3}\left(1 - \frac{\alpha+\gamma-1}{2^\kappa}\right)^{n\alpha}$.*

The proof of Theorem 1 is given in Appendix A.

6 Composite Signatures and Cryptocurrencies

As discussed earlier, composite signatures can be used to enhance anonymity in cryptocurrencies (such as bitcoin) by unlinking the input and output addresses from where funds move. We summarize the ideas below.

In bitcoin transactions, the sending addresses (i.e., public keys) are linked to the other sending addresses and receiving addresses in a transaction. This link is 'hard' in the sense that it provides a cryptographic proof of funds transfer between those addresses. For example, suppose owner of address pk_1 wants to

transfer 1 bitcoin to address pk_2. The transaction will be the message **"Take 1 bitcoin from pk_1; Put 1 bitcoin in pk_2"**, signed under the public key pk_1. This transaction cryptographically links the addresses pk_1 and pk_2. The owner of pk_1 cannot later deny sending the funds to pk_2.

Composite signatures enable significantly higher anonymity by removing linkages from sending and receiving addresses. This allows senders to release funds without referring to receiving addresses or other sending addresses. Using composite signatures, the transaction in the above example will consist of two messages (1) the message **"Take 1 bitcoin from pk_1"** signed under pk_1, and (2) the message **"Put 1 bitcoin in pk_2"** signed under a randomly generated public key (which we call the masking key). The two signatures will then be combined into one composite signature and broadcast to the network. Other peers may add more signatures from their transactions before broadcasting further (to increase unlinkability). Since individual signatures in a composite signature cannot be extracted, the composite signature serves as a secure record of the transaction, despite the fact that messages do not contain references to other public keys. Senders can claim *plausible deniability*, since the composite signature cannot serve as proofs of knowledge of the receiving addresses.

We proposed the use of 'joiners', 'merging services' and the knapsack problem [20] to further increase plausible deniability.

Anti-censorship and participation incentive: The proposal also has a desirable anti-censorship property. Because composite spends cannot be de-aggregated, if a miner learns of a 'desirable' spend only after it is aggregated with censored spends, it must take all or none. The prospect of open aggregates also allows relaying nodes to participate in collecting transaction fees.

7 Integrating with Bitcoin

The modified transactions described here use composite signatures instead of ordinary signatures (such as ECDSA). Their construction uses bilinear pairings on elliptic curves. Some examples of such pairings are: Weil pairing [22], the Tate-Lichtenbaum pairing [23] and the Eta Pairing [24].

Efficiency: Public keys are elements of G_1, which are elements of a suitable finite field. Based on [25, 26], such elements can be represented in about 30 bytes for 128 bits of security. The signatures constitute one group element and n κ-bit strings (the random rs). The size of signatures increases linearly. Below we consider the possibility of using a weaker scheme where these rs are removed. Signature verification requires several pairing computations, which can be performed fairly efficiently [25, 26] (< 10 ms on a Pentium).

Increasing efficiency: Our composite signature construction extends the aggregate signatures of [4] by including a random string r in the signature. The signatures of [4] are constant-size (about 30 bytes) because the r is not included.

However, they do not satisfy the security of Def. 2. In practice, however, a weaker security notion is sufficient. In the weaker notion we require the forgery ℓ_A to be not weakly signable (Def. 1). We posit that the construction of [4] is secure in this weaker sense. Furthermore, for our application, an even weaker form of security - the non-adaptive case - should be sufficient. This requires the adversary to output a forgery after making only one sign query. The signatures of [4] satisfy this model [5]. Therefore, we envisage the construction of [4] to be used in our application.

Based on above parameters, transaction size is comparable to that in the existing protocol. In order to verify transactions/blocks created via composite signatures, all relevant masking keys need to be available. These can either be part of the payload or kept in a publicly database (with hashes as payloads). The remaining details of the protocol (such as pruning, etc) remain the same.

The modified protocol can possibly co-exist with the current protocol. We add the new type of transaction output based on composite signatures. These outputs can be mixed with standard outputs. A composite signature based output will be spent using the new protocol described here. A transaction can even be constructed using a mix of these outputs. We leave this as further work.

8 Conclusion and Future Work

Bitcoin is a popular peer-to-peer cryptocurrency with a weak form of anonymity. We presented an enhancement of the bitcoin protocol to increase anonymity. Our method is based on a new cryptographic primitive known as *composite signatures*. Composite signatures are an extension of Boneh et al.'s aggregate signatures [4] and have the property that multiple signatures can be aggregated into one signatures such that once aggregated, the individual signatures cannot be recovered. We gave the security model of composite signatures and presented a construction with a security proof under the random oracle model and the computational Diffie-Hellman assumption in bilinear maps. We also presented a weaker notion of composite signatures (Def. 1), which may be interesting because the publicly computable signatures exhibit a group structure.

Composite signatures can be used to enhance anonymity in cryptocurrencies such as bitcoin by unlinking the input and output addresses from where funds move. In the current bitcoin protocol the sending addresses are linked to the other sending addresses and receiving addresses in a transaction. This link is 'hard' in the sense that it provides a cryptographic proof of funds transfer between those addresses. We use composite signatures to remove all linkages from the sending and receiving addresses. This enables senders to sign messages releasing funds without mentioning the receiving addresses or other sending addresses, thereby providing plausible deniability. Additionally, several transactions can be combined into one large transaction (possibly via the knapsack problem) in order to further obfuscate the links.

Acknowledgment: Thank to the anonymous reviewers for their feedback and to Gregory Maxwell for pointing out the anti-censorship and participation incentive properties of the proposed method.

References

1. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System.
2. Sergio Martins and Yang Yang. Introduction to bitcoins: a pseudo-anonymous electronic currency system. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '11, pages 349–350, Riverton, NJ, USA, 2011. IBM Corp.
3. Wenbo Mao and Kenneth G. Paterson. On the plausible deniability feature of internet protocols. `http://isg.rhul.ac.uk/~kp/IKE.ps`, 2002.
4. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.
5. Jean-Sébastien Coron and David Naccache. Boneh et al.'s k-element aggregate extraction assumption is equivalent to the Diffie-Hellman assumption. In *ASIACRYPT*, pages 392–397, 2003.
6. Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *EUROCRYPT*, pages 74–90, 2004.
7. Huafei Zhu, Feng Bao, Tieyan Li, and Yongdong Wu. Sequential aggregate signatures for wireless routing protocols. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 4, pages 2436–2439 Vol. 4, 2005.
8. Di Ma. Practical forward secure sequential aggregate signatures. In Masayuki Abe and Virgil D. Gligor, editors, *ASIACCS*, pages 341–352. ACM, 2008.
9. Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 276–285, New York, NY, USA, 2007. ACM.
10. Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, pages 465–485, 2006.
11. Marc Fischlin, Anja Lehmann, and Dominique Schrder. History-free sequential aggregate signatures. In Ivan Visconti and Roberto Prisco, editors, *Security and Cryptography for Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 113–130. Springer Berlin Heidelberg, 2012.
12. Kyle Brogle, Sharon Goldberg, and Leonid Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 644–662. Springer Berlin Heidelberg, 2012.
13. E. Androulaki, G. Karame, M. Roeschlin, T. Scherer, and S Capkun. Evaluating user privacy in bitcoin. Cryptology ePrint Archive, Report 2012/596, 2012.
14. Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In Yaniv Altshuler, Yuval Elovici, Armin B. Cremers, Nadav Aharony, and Alex Pentland, editors, *Security and Privacy in Social Networks*, pages 197–223. Springer New York, 2013.

15. Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. Cryptology ePrint Archive, Report 2012/584, 2012. `http://eprint.iacr.org/`.
16. Zerocoin: Anonymous distributed e-cash from bitcoin, 2012.
17. Gregory Maxwell. Coinjoin: Bitcoin privacy for the real world, 2013.
18. Bitcoin Developers. Bitcoin client source code (github), 2008.
19. David Pisinger. Where are the hard knapsack problems. *Computers and Operations Research*, 32, 2005.
20. Vasek Chvatal. Hard knapsack problems. *Operations Research*, 28(6):1402–1411, 1980.
21. Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, December 2007. Prelim. in FOCS 2002.
22. Victor S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.
23. Yukihiro Uchida and Shigenori Uchiyama. The Tate-Lichtenbaum pairing on a hyperelliptic curve via hyperelliptic nets. In Michel Abdalla and Tanja Lange, editors, *Pairing-Based Cryptography Pairing 2012*, volume 7708 of *Lecture Notes in Computer Science*, pages 218–233. Springer Berlin Heidelberg, 2013.
24. F. Hess, N.P. Smart, and F. Vercauteren. The Eta pairing revisited. *Information Theory, IEEE Transactions on*, 52(10):4595–4602, 2006.
25. Naoyuki Shinohara, Takeshi Shimoyama, Takuya Hayashi, and Tsuyoshi Takagi. Key length estimation of pairing-based cryptosystems using nT pairing. In *Proceedings of the 8th international conference on Information Security Practice and Experience*, ISPEC'12, pages 228–244, Berlin, Heidelberg, 2012. Springer-Verlag.
26. Michael Scott. Scaling security in pairing-based protocols. *IACR Cryptology ePrint Archive*, 2005:139, 2005.

A Proof of Theorem 1

Proof. Let $g, g^x, g^y \in G_1$ be the given CDH instance we need to solve (our goal is to compute g^{xy}). We show how to solve this using A as a black-box.

Setup: We generate $a_1, a_2, \ldots a_n \xleftarrow{R} \mathbb{Z}_q$ and set the target public keys as $pk_i = g^{x+a_i}$ for $1 \leq i \leq n$. The set $PK = \{pk_i\}_{i\in[1..n]}$ is given to A.

H-list: A can query the random oracle H on points from $\Sigma^* \times \Sigma^\kappa \times G_1$. To respond to such queries, we maintain a list called the H-list, which is initially empty and contains tuples of the type

$$(m, r, pk, h, b, c, d) \in \Sigma^* \times \Sigma^\kappa \times G_1 \times G_1 \times \mathbb{Z}_q \times \mathbb{Z}_2 \times \pm 1,$$

such that $h = g^{cdy+b}$ always holds.

H-Queries: On $H(m_i, r_i, pk_i)$ query, if a tuple $(m_i, r_i, pk_i, h_i, b_i, c_i, d_i)$ exists in the H-list, we respond with $h_i = H(m_i, r_i, pk_i)$, otherwise we add such an entry as follows. We generate $b_i \xleftarrow{R} \mathbb{Z}_q$ uniformly and set $d_i = 1$. If $pk_i \notin PK$, we set $c_i = 0$, otherwise we set $c_i = 1$. Finally we set $h_i = g^{c_i y + b_i}$ and respond with $h_i = H(m_i, r_i, pk_i)$. In effect, if $pk_i \notin PK$, we set $h_i = g^{b_i}$, otherwise we set $h_i = g^{b_i+y}$.

Sign queries: Let $\ell = ((m_1, pk_1), (m_2, pk_2), \ldots (m_k, pk_k))$ be any sign query for $k \leq n$. To respond to this, we generate k random numbers $r_1, r_2, \ldots r_k \stackrel{R}{\leftarrow} \Sigma^\kappa$ and for each $i \in [1..k]$ we check the H-list for entries starting with (m_i, r_i, pk_i). If any such entry exists, we report failure and abort, otherwise we add the entries as follows. We uniformly select k pairs $((c_1, d_1), (c_2, d_2), \ldots (c_k, d_k)) \in (\mathbb{Z}_2 \times \pm 1)^k$ such that $\sum_{i=1}^k c_i d_i = 0$ and $k - \sum_{i=1}^k c_i \in \mathbb{Z}_2$. The latter says that at most one of the c_is can be 0.[5] We then generate $b_1, b_2, \ldots b_k \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and for each $i \in [1..k]$, we set $h_i = g^{c_i d_i y_i + b_i}$. We add $(m_i, r_i, pk_i, h_i, b_i, c_i, d_i)$ to the H-list.

Let $\sigma' = g^{\sum_{i=1}^k (x+a_i)(c_i d_i y + b_i)} = g^{xy \sum_{i=1}^k c_i d_i + \sum_{i=1}^k x b_i + a_i c_i d_i y + a_i b_i}$. We know that $\sum_{i=1}^k c_i d_i = 0$ (by construction). Therefore, $\sigma' = g^{\sum_{i=1}^k x b_i + a_i c_i d_i y + a_i b_i}$, a value that can be computed by us. Also, $\sigma = (\sigma', \{r_1, r_2, \ldots r_k\})$ is a valid signature on ℓ, which is our response to the query.

Output: Finally, A outputs a pair (σ_A, ℓ_A). If σ_A is not a valid forgery on ℓ_A, we report failure. Let PK_A be the set of public keys in this forgery. Some of these keys may not be from PK. Let $PK^\# = PK_A \backslash PK$ and $PK^* = PK \cap PK_A$.

By construction, all c_is in the H-list corresponding to the messages signed under $PK^\#$ are 0. Therefore, the respective b_is are the discrete logarithms (to base g) of the corresponding h_is. Hence, we can compute the sub-composite signature corresponding to the messages of PK^*, denoted by σ_* (we compute this by first computing the sub-composite signature corresponding to the messages of $PK^\#$ and "dividing" σ_A by that).

Let $((a_1^*, b_1^*, c_1^*, d_1^*), \ldots, (a_{k^*}^*, b_{k^*}^*, c_{k^*}^*, d_{k^*}^*))$ be tuples containing a_is and H-list entries corresponding to PK^*. If $\sum_{i=1}^{k^*} c_i^* d_i^* = 0$, we report failure and abort, otherwise σ_* corresponds to a signature we could not have computed ourselves, which can be used to solve the CDH problem as follows. We know that $\sigma_* = (\sigma'_*, \{r_1^*, \ldots r_{k^*}^*\})$ such that $\sigma'_* = g^{\sum_{i=1}^{k^*} (x+a_i^*)(c_i^* d_i^* y + b_i^*)} = g^{xy \sum_{i=1}^{k^*} c_i^* d_i^*} \cdot g^{\sum_{i=1}^{k^*} x b_i^* + a_i^* c_i^* d_i^* y + a_i^* b_i^*} = g^{xyz} \cdot w$ for some nonzero w and z that we know. Using this, we can compute $g^{xy} = (\sigma'_*/w)^{1/z}$.

It now remains to bound the probability of success. Define events:

- $\mathcal{E}_1$ = We do not abort during sign queries.
- $\mathcal{E}_2 = \mathcal{E}_1$ and A outputs a successful forgery.
- $\mathcal{E}_3 = \mathcal{E}_2$ and $\sum_{i=1}^{k^*} c_i^* d_i^* \neq 0$.

Then $\Pr[\text{success}] = \Pr[\mathcal{E}_3 | \mathcal{E}_2] \cdot \Pr[\mathcal{E}_2 | \mathcal{E}_1] \cdot \Pr[\mathcal{E}_1]$.

Claim 1 $\Pr[\mathcal{E}_1] \geq \left(1 - \frac{\alpha+\gamma-1}{2^\kappa}\right)^{n\alpha}$

Proof. Consider the number of entries in the H-list corresponding to a given (message, public-key) pair (m, pk). Each H-query can add at most one entry to the H-list for this pair. Since a sign query can contain at most one instance of the pair (m, pk), therefore, each sign query can add at most one entry in the

[5] These pairs can be generated as follows. First set all c_is to 1. If k is odd, randomly set one of the c_is to 0. Then for those c_is that are 1, randomly set half of the d_is to $+1$ and the rest to -1.

H-list for this pair. Therefore there can be a maximum of $\alpha+\gamma-1$ entries in the H-list corresponding to (m, pk). Now select $r \xleftarrow{R} \Sigma^\kappa$ and consider the event that an entry beginning with (m, r, pk) exists in the H-list. Since there are 2^κ possible ways to select r, we can be assured that $\Pr[\text{no entry in H-list for } (m, r, pk)] \geq 1 - \frac{\alpha+\gamma-1}{2^\kappa}$. Now there can be maximum n pairs in a sign query. Therefore, $\Pr[\text{we do not abort in one sign query}] \geq \left(1 - \frac{\alpha+\gamma-1}{2^\kappa}\right)^n$, and so

$$\Pr[\mathcal{E}_1] = \Pr[\text{we do not abort in } \alpha \text{ sign queries}] \geq \left(1 - \frac{\alpha+\gamma-1}{2^\kappa}\right)^{n\alpha} \qquad \square$$

Claim 2 $\Pr[\mathcal{E}_2|\mathcal{E}_1] = \epsilon$.

Proof. If we do not abort during sign queries, then the view of the adversary is identical to a real simulation, and it follows that $\Pr[\mathcal{E}_2|\mathcal{E}_1] = \epsilon$. $\square$

Claim 3 $\Pr[\mathcal{E}_3|\mathcal{E}_2] \geq 1/3$

Proof. Split H-list entries into two disjoint sets based on how they are generated:

1. S_1: Sign queries on single (message, public-key) pairs. Here $\Pr[c = 0] = 1$.
2. S_2: H-queries or sign queries on two or more (message, public-key) pairs. It can be checked that $\Pr[c = 0] \leq 1/3$ for such entries.

Let the forgery contain k^* (message, public-key) pairs. Let $\{(m_i^*, r_i^*, pk_i^*)\}_{i\in[1..k^*]}$ be the set of tuples corresponding to the forgery. We ensure that an entry for each tuple exists in the H-list (by simulating H-queries ourselves if necessary).

Lemma 1. *If the forgery is valid (i.e., ℓ_A is not signable), then at least one of the tuples in the forgery must must correspond to an element of S_2.*

Proof. If all tuples $\{(m_i^*, r_i^*, pk_i^*)\}_{i\in[1..k^*]}$ in the forgery correspond to elements from S_1, then A made sign queries on every pair (m_i^*, pk_i^*), possibly more than once. By definition, ℓ_A is signable. Hence the forgery cannot be valid. $\square$

For any signature σ_ℓ from the sign queries or the forgery, define $f(\sigma_\ell) = \sum_{i=1}^k c_i d_i$, obtained from corresponding entries $(m_i, r_i, pk_i, h_i, b_i, c_i, d_i)$ in the H-list. A's goal is to maximize $\Pr[\neg\mathcal{E}_3|\mathcal{E}_2] = \Pr[f(\sigma_*) = 0]$.

Since we did not abort during the sign queries, each tuple (m_i^*, r_i^*, pk_i^*) was used in at most one sign query. Therefore A's view of any of the c_i^*s for tuples from S_2 is independent of any queries. Extending Lemma 1, we can see that if ℓ_A is not signable, then A's view of $f(\sigma_*)$ is independent of all queries. An upper bound for $\Pr[\neg\mathcal{E}_3|\mathcal{E}_2]$ then gives us the worst case scenario.

Keeping tuples from S_1 in the forgery is not useful for A, since $c_i = 0$ for such values and so $f(\sigma_*)$ is independent of them. Therefore, assume that A's forgery contains only elements from S_2. Now S_2 can be further divided into: (1) S_2' consisting of entries due to H-queries and (2) S_2'' consisting of entries due to sign queries. Since for elements of S_2'', the d_is are uniformly distributed between ± 1, while for those of S_2', the d_is are guaranteed to be $+1$, a symmetric

argument shows that including elements from S_2' is not beneficial to A since it only biases $f(\sigma_*)$ towards nonzero. Therefore, assume that A's forgery contains only elements from S_2''. A counting argument shows that if all elements are from S_2'', then $\Pr[f(\sigma_*) = 0] \leq 2/3$, with the maximum occurring when A extracts a 2-tuple signature from a 4-tuple signature. Hence $\Pr[\mathcal{E}_3|\mathcal{E}_2] \geq 1/3$ □

This proves Theorem 1. □