SendProof Protocol v1.0

The Bytecoin Developers

July 31, 2018

Abstract

We propose a zero-knowledge protocol which can be used to prove money transfers for the CryptoNote-based currencies. A single transaction can involve several coin recipients. The SendProof signature can be generated by the sender selectively for any subset of recipients without compromising in any way the anonymity of other recipients or other transactions. The signature can be checked non-interactively by any third party receiving it. It can be generated retroactively for any past transactions given only the recipient's public address. The protocol is implemented in the Bytecoin system and is available via the daemon API as well as the GUI desktop wallet.

Contents

1 Introduction

Financial transactions always require a kind of verification so that a third party can confirm that they have happened. However the pseudonymous nature of cryptocurrencies makes such verifications difficult. The problem, in particular, is significant for cryptocurrencies with anonymity based on the CryptoNote [S] protocol which we consider in this article. This protocol is focused on the total anonymity of both the senders and the receivers for all transactions, hiding their identities via a version of ring signatures

and making it impossible for a random observer to identify the involved parties. In the case of some dispute the sender could, in principle, publish their private transaction keys, however once their secret is disclosed they no longer have any way to prove the ownership of the transaction. Anyone intercepting the keys could also publish them as their own and claim to be the sender. Publishing private keys over a secure channel is still undesirable since it can lead to de-anonymization of the other transaction recipients.

A better solution would be a zero-knowledge proof of the sender's identity which could be verified by any third party without disclosing any extra private information. One would also desire a proof of transaction ownership which wouldn't require sender's de-anonymization, i.e. without publishing the sender's secrets such as wallet keys.

The SendProof protocol, implemented in the Bytecoin [BCN] cryptocurrency, provides one possible solution to this problem. With the SendProof protocol the sender can provide a proof of sending given amount of money to given receiver's address in a particular transaction. The proof is non-interactive and can be verified at any time by any third party that gets the proof.

Here are some example use cases where SendProofs should help:

- Customers performing anonymous transactions to a shop can use the SendProofs to protect themselves from possible fraud since they have an irrefutable proof of purchase.

- Similarly, a shop can protect itself from possible misplaced claims of purchase from the customers. A real purchaser can always produce a valid SendProof signature.

- SendProofs offer a fine-grained level of control over transaction sharing. For example, a client can deposit money in the bank with a condition that a fixed interest would be transferred to a demand account. The bank can provide SendProofs which confirm these interest transactions without disclosing any other operations with the money.

- Another version of the case above is a blockchain lottery. The lottery organizers can prove that they have made a payout to one of the participants by providing the relevant SendProof.

- The basic CryptoNote protocol doesn't provide any convenient way of sender identification. In the past this was a problem for currency exchanges. The commonly adopted solution nowadays is to create multiple wallets at the exchange, with (at least one) address for each client and use it as a natural identifier. But SendProofs provide a possible alternative: multiple clients sending money to the same account can identify their transactions to an exchange by providing the relevant SendProofs with their credentials.

While the actual implementation is based on a group of points of the specific elliptic curve Curve25519 [B1, B2], both the CryptoNote and the SendProof protocols work for any cyclic group. The main restriction, as usual, is that the discrete logarithm problem *(DLP)* in the group is infeasible to solve, which means that it is extremely unlikely to find any nontrivial relations among random group elements within a reasonable time.

2 Preliminaries

We operate under the standard assumptions that the attack on the protocol is performed by a probabilistic polynomial time adversary. The construction and proofs can be regarded as a special case of a more general system described in [CS].

Let $\mathbb{G}$ be a finite cyclic group of prime order ℓ such that the DLP in it is infeasible to solve. For example, $\mathbb{G}$ could be the multiplicative group of some Galois field [MM] $\mathbb{F}_q$, $q = p^n$ for p a prime and n a positive integer. Another example comes from the theory of elliptic curves [Sil], the simplest examples of which are equations of the form $E : y^2 = x^3 + Ax + B$ over some field $\mathbb{K}$. The solutions (x, y) of this equation, together with the unique "point at infinity" (∞, ∞), form an abelian group denoted by $E(\mathbb{K})$, which is always finite for $\mathbb{K} = \mathbb{F}_q$. If $E(\mathbb{K})$ has a large cyclic subgroup $\mathbb{G}$ of prime order, then the DLP in $\mathbb{G}$ is often infeasible with contemporary methods.

We fix a generator $\mathcal{G} \in G$. For any element $S \in \mathbb{G}$ and any field element $n \in \mathbb{F}_\ell$ the n-fold product of S with itself will be denoted by nS, using the additive notation for the group multiplication.

We will denote the elements of $\mathbb{F}_l$ by small Latin letters and call them *private keys* (e.g. $a, b, p \in \mathbb{F}_\ell$), while the corresponding group elements will be denoted by capital Latin letters and called *public keys* (e.g. $A = a\mathcal{G}$).

We will also require two hash functions: the scalar hash function $\mathcal{H}_s : \{0,1\}^* \to \mathbb{F}_\ell$ which maps bit strings to integers mod ℓ, and the point hash function $\mathcal{H}_\mathbb{G} : \mathbb{G} \to \mathbb{G}$ which maps a group element to another pseudo-random group element.

We assume that the Diffie–Hellman problem in $\mathbb{G}$ is infeasible to solve, which means that given an element $X \in \mathbb{G}$ and, for a pair $u, v \in \mathbb{Z}$, a pair of elements $uX, vX \in \mathbb{G}$, it is infeasible to find uvX if u and v are unknown. This problem underlies the Diffie–Hellman key exchange protocol and is required in CryptoNote for the unlinkability of transaction outputs.

3 The CryptoNote Protocol

We recall briefly the outline of the protocol. For further details, as well as the proofs of the protocol's security, consult the original article [S].

The CryptoNote protocol operates with individual "coins", which are triplets $(amount, p, P)$ where *amount* is the coin denomination, p is the *coin's private key* and P is the corresponding public key. The private key is known only to the coin's owner and is required to spend it: during the transaction a person is required to correctly compute the public key

$$I = p \cdot \mathcal{H}_{\mathbb{G}}(P)$$

called the *key image* for each input coin, as well as to provide a valid signature which depends on all input coins' p and I (but does not leak any information about p). Only one transaction input with each specific value of the key image can exist in the blockchain, and the protocol guarantees that only the unique correct key image can be accepted for each coin, meaning that double spending is impossible. Theft and fraud are also impossible since the value of p cannot be computed from I and P even if all other peers in the coin network are malicious.

To perform a transaction, the sender chooses a sequence of owned coins (called *inputs*) which will be spent in the transaction, and produces a sequence of new coins (called *outputs*) with varying amounts and receivers (possibly themselves). Logically, the transaction proceeds in two steps: the verification phase where a person proves that they have the right to spend the coin, and the spending phase where new outputs are generated.

In the verification phase the sender produces a linkable ring signature for each owned coin $(amount, p, P)$. The purpose of the signature is twofold: it proves that the sender knows the coin's private key p without revealing its value, while also concealing the sender's identity and the exact coin spent. To do this, the sender randomly selects a sequence of coins $P_1, \ldots, P_n$ of equal denomination, with one of them being the coin P that is spent. The ring signature consists of the key image I together with two vectors

$$c_1, \ldots, c_n, r_1, \ldots, r_n \in \mathbb{F}_\ell^n$$

and the signature is accepted if and only if

$$\sum_i c_i = \mathcal{H}_s\left(m, L'_1, \ldots, L'_n, R'_1, \ldots, R'_n\right) \mod l$$

where m is the extra signature message string and

$$\begin{aligned} L'_i &= r_i \mathcal{G} + c_i P_i \\ R'_i &= r_i \mathcal{H}_{\mathbb{G}}(P_i) + c_i I \end{aligned}$$

A valid signature guarantees that one of the coins in the sequence was indeed spent by the sender but does not reveal any extra information. If the sender attempted to double-spend the coin, then they would be forced to produce

the ring signature with the duplicate key image, which would be rejected by the consensus algorithm.

During the spending phase the sender picks a random number $r \in \mathbb{F}_l$ *(transaction private key)* and creates a new coin for each possible output with receiver (a, b, A, B), where a, b are the receiver's private keys and A, B are the respective public keys. The denominations of new coins are arbitrary[1] with the only restrictions being that they are non-negative (so no money can be created) and that the sum of the input amounts is greater than or equal to the sum of the output amounts (a tiny amount can be withheld as part of the miners' fee). For each new coin its public key is generated as follows: the sender computes two public keys

$$R = r\mathcal{G}$$
$$D = rA$$

R is called the transaction public key. The public key of the new coin is defined as

$$P = \mathcal{H}_s\left(i \| D\right)\mathcal{G} + B$$

where i is the index of the corresponding output (known to everyone and a property of the coin). The ring signatures in the verification phase are used to sign the data of all output coins.

Since $B = b\mathcal{G}$, the private key of the coin is equal to $p = \mathcal{H}_s\left(i \| D\right) + b$, but no one except the coin's receiver can compute it. The sender commits the public transaction key R, the tuple $(amount, P)$ of each output coin and the ring signatures signing the entire transaction to the blockchain. The value of D normally isn't published since it would allow to identify the coin's receiver. The receiver can compute the value of D as $D = aR$ and find their coin by checking all coins' public keys against the value $\mathcal{H}_s\left(i \| aR\right)\mathcal{G} + B$.

4 The SendProof Protocol

Let's say Sandra sends money to Reece and wants to prove it to Victor. The simplest way would be to publish the transaction private key r or the value $D = rA$ which only Sandra knows. However, the problem is that it is difficult to prove that Sandra really was the first to own that data, especially if some data corruption or malicious group intercepts her messages and deletes her posts. Once the value becomes public, anyone can claim to have it beforehand. Even pre-publishing the fingerprints in advance on trusted third party servers can be potentially insufficient, and for time-critical transaction disputes it may be impossible. Sandra may also want to conceal other

[1] In practice, denominations are usually chosen by an algorithm optimizing the network efficiency.

recipients of the transaction, even if that other recipient is just herself receiving the unspent change from her transaction: public reveal of her owned coins could lead to some major privacy or security risk in the future. A better option would be to publish a zero-knowledge proof of knowledge for the transaction private key which would guarantee its existence without leaking the value. If this proof can contain an extra message, then Sandra can incorporate her identifying information (like forum name or public keys) into the proof. Even if no identifying information is available, she could still prove her ownership to Victor by encoding in the proof some private message supplied by him. The SendProof protocol is the implementation of such zero-knowledge proofs for CryptoNote–based currencies [CNC].

To generate a signature Sandra chooses a random blinding number $k \in \mathbb{F}_\ell$, a message m and computes the following values:

$$\begin{aligned} D &= rA \\ h &= \mathcal{H}_s\left(m \| k\mathcal{G} \| kA \| D \| R \| A\right) \\ t &= k - rh \end{aligned}$$

The tuple (h, t, D, R, A, m) is the published signature.

To verify the signature Victor computes the following values:

$$\begin{aligned} X &= t\mathcal{G} + hR \\ Y &= tA + hD \\ h' &= \mathcal{H}_s\left(m \| X \| Y \| D \| R \| A\right) \end{aligned}$$

He accepts the signature if $h = h'$, otherwise rejects it.

A proper zero-knowledge signature scheme should have three properties:

Completeness. The correct signature is always accepted.

Soundness. A third party which doesn't know the secret can produce a valid signature only with a vanishingly small probability.

Zero-knowledge. A person knowing the signature cannot get any information apart from the fact that Sandra knows the secret.

SendProofs actually satisfy a criterion which is stronger than simple soundness and common for cryptocurrencies, since both the transaction and the proof are provided by the same party.

Strong soundness. A probabilistic polynomial time adversary cannot produce a fake transaction and a proof that it is valid, i.e. almost certainly if a PP adversary produces a tuple (R, D, A, t, h), then they know r such that $D = rA$, $R = r\mathcal{G}$.

Note that if the protocol didn't have strong soundness, then Sandra could publish a transaction which Reece couldn't find and spend. However in practice this isn't a problem for CryptoNote since Reece only needs the value of D to find the transaction and compute the private key, so he will be able to claim it as soon as he gets the SendProof signature, even if the proof or the transaction is malformed. Anyone can easily verify the correctness of D by simply scanning the blockchain and comparing all public coin keys with $\mathcal{H}_s(D)\mathcal{G} + B$, the rest of the SendProof signature is really required to verify the transaction ownership.

We will discuss soundness and zero-knowledge in Security of the Protocol. To see that a valid signature is always accepted, note that in this case

$$\begin{aligned} X &= (k - rh)\mathcal{G} + hR \\ &= k\mathcal{G} \end{aligned}$$

since $R = r\mathcal{G}$. Similarly,

$$\begin{aligned} Y &= (k - rh)A + hD \\ &= kA \end{aligned}$$

The value of D is sufficient to find the transaction to Reece, but the transaction private key itself isn't leaked. The protocol doesn't require any identifying information from Sandra other than the one she explicitly decides to include in the message field (e.g. her wallet's address and coin's private and public keys are not required). The proof requires Sandra to remember the value of r from the specific transaction, which leaves her with three options:

1. produce the SendProof immediately after the transaction without any possibility to provide it later;
2. store all transaction keys in a secure environment;
3. use a deterministic pseudo-random generator to choose each private key, with the ability to reconstruct them at any point in time from the secret seed.

The third option has a potential security problem, since if a third party learns the pseudo-random seed they can learn the private keys of all Sandra's transactions and track their amounts and recipients. However, the seed isn't published anywhere through the normal CryptoNote protocols and should be secure unless Sandra publishes it herself or her devices are compromised. On the other hand, it is by far the most convenient option since one can never know which transactions could be disputed, and unless Sandra uses the option 2 and stores all private transaction keys she can never prove her ownership with certainty. Storing all private keys securely

is more difficult and no more secure than storing a single key seed, thus the choice is really between options 1 and 3. Option 3 is implemented in Bytecoin as a reasonable compromise between usefulness and security.

Note that publishing a SendProof for a single output of Reece in the transaction also discloses all of his other outputs, since they share the value of D and the coin indices are public. This means that if one sends several coins to the receiver which aren't all logically part of a single financial transaction (different coins in a single payment), then one should split them into different transactions with different transaction keys. Any other receiver included in the same transaction will keep their identity anonymous unless the sender publishes their specific SendProof. This is part of the zero-knowledge property of the protocol which we will discuss later.

5 Security of the Protocol

We prove the security of the protocol in the *random oracle model*, where the hash function is assumed to be a random function from strings to the finite field $\mathbb{F}_\ell$. The proof is based on the forking lemma [PS], which we reproduce here for convenience.

Assume that the signature has the form $(m, \sigma_1, h, \sigma_2)$ where m is an arbitrary extra message string, σ_1 is the part of the signature which doesn't depend on the random oracle, σ_2 is the part that depends on it, and $h = \mathcal{H}_s(m, \sigma_1)$ is the value of the random oracle on the pair (m, σ_1). We will usually omit the message field from the signature notation.

Lemma 1 (the forking lemma, [PS])**.** *Let $\mathcal{A}$ be a PP Turing machine, given only the public data as input. If $\mathcal{A}$ can find, with non-negligible probability, a valid signature $(m, \sigma_1, h, \sigma_2)$, then, with non-negligible probability, a replay of this machine, with the same random tape and a different oracle, outputs two valid signatures $(m, \sigma_1, h, \sigma_2)$ and $(m, \sigma_1, h', \sigma_2')$ such that $h \neq h'$.*

In the statement of the lemma the probability is taken over all tapes, oracles, messages, and keys. Intuitively, it means that the attack cannot depend in any way on the specificity of the random oracle function and should produce a valid signature regardless of the specific oracle value. It also shows that the random oracle call acts as a checkpoint in any signature algorithm, separating all parts of the signature into the fixed preconditions and the postconditions which cannot cannot be satisfied without knowing the random oracle response. The forking lemma thus shows that a non-interactive proof with a random oracle has the same security as an interactive proof where the prover first commits all preconditions to the verifier and only then receives an arbitrary verifier-chosen number as a challenge to satisfy.

Proposition 2 (strong soundness)**.** *If a PP attacker can produce a valid signature $\sigma = (R, D, A, h, t)$ with non-negligible probability, then there is a value $r \in \mathbb{F}_\ell$ such that $R = r\mathcal{G}$ and $D = rA$.*

In this case $P = \mathcal{H}_s(D)\,\mathcal{G} + B$ is a valid public coin key owned by the receiver with the public key pair (A, B).

Proof. The forking lemma applies to the SendProof protocol with

$$\sigma_1 = (R, A, D, X, Y)$$

and $\sigma_2 = t$. If an attacker can produce a valid signature with non-negligible probability, then, with non-negligible probability, we can find two different oracles $\mathcal{H}_s(\cdot)_1$ and $\mathcal{H}_s(\cdot)_2$ such that

$$\begin{aligned} h_{1,2} &= \mathcal{H}_s(\sigma_1)_{1,2} \\ h_1 &\neq h_2 \end{aligned}$$

and the values of σ_1 are the same. These two different values of the oracle will result in two valid signatures $t_{1,2}$, such that

$$\begin{aligned} X &= t_{1,2}\mathcal{G} + h_{1,2}R \\ Y &= t_{1,2}A + h_{1,2}D \end{aligned}$$

The last two equations imply that

$$\begin{aligned} (t_1 - t_2)\,\mathcal{G} &= (h_2 - h_1)\,R \\ (t_1 - t_2)\,A &= (h_2 - h_1)\,D \end{aligned}$$

which shows that $r = \frac{t_1 - t_2}{h_2 - h_1}$. □

Corollary 3 (unforgeability)**.** *If the DLP for the group $\mathbb{G}$ is unsolvable for a PP attacker and he doesn't know the private transaction key, then he can produce a valid signature $\sigma = (R, D, A, h, t)$ only with a negligible probability, where the probability is counted over all public transaction key pairs (R, A), tapes and oracles.*

Proof. From the proof of strong soundness we see that if an attacker can produce a valid signature with a non-negligible probability, then there is a non-negligible probability that they can compute the value r such that $R = r\mathcal{G}$. Since r can be arbitrary, it means that with non-negligible probability they can solve the DPL in $\mathbb{G}$ by considering arbitrary public keys A and random oracles (by adding arbitrary messages m to the signature we can associate a family of random oracles to each single one) and forging the SendProof for the pair (R, A). □

In the following we consider the known SendProof signatures for different messages and coin outputs as different values of the *signing oracle*. A signing oracle is a function that produces a valid signature for any tuple (m, R, A) which is assumed as publicly available in the theorem statement (note that the SendProof signature depends only on the transaction keys and the recipient view keys, and not on any extra data like the output coin keys or ring signatures). A polynomial adversary can make at most a polynomial number of calls to this oracle, which is another way of talking about a polynomial (in the security parameter) number of available valid signatures. Note that the signing oracle itself depends on the random oracle, however if it follows the signature scheme (which we assume it does as we consider it to be an honest prover) it can only depend on the value of the random oracle at the specific point used in the protocol. In particular, one can change the values of the random oracle used in one signature without affecting any other signatures.

Proposition 4. *The SendProof protocol doesn't leak any information about private keys in the following sense:*

1. *Assume that for a given public transaction key R the attacker can access its signing oracle, then he can deduce its private transaction key only with a negligible probability.*

2. *Assume that the attacker has access to a signing oracle which produces a valid SendProof signature for any tuple (m, R, A), except for a chosen tuple (m_0, R_0, A_0). In this case the attacker can produce a valid signature for (m_0, R_0, A_0) only with a negligible probability.*

3. *Assume that the attacker knows a polynomial number of valid SendProof signatures for a single output coin (R, A) with different messages $m_{1,\dots,n}$, then he can produce only with a negligible probability a valid signature for a different message $m \neq m_{1,\dots,n}$.*

4. *Assume that a polynomial number of different recipients $(A_i, B_i)_{i=0,\dots,n}$ receive output coins from the same transaction with the same transaction key pair (r, R). If the attacker knows the SendProofs for all but one recipients $(A_i, B_i)_{i=1,\dots,n}$ $(i \neq 0)$, then he can produce a valid signature for the remaining recipient (A_0, B_0) only with a negligible probability.*

5. (unlinkability) *Under the assumptions of 4, assume that the attacker doesn't know (A_0, B_0). Assume that the Diffie–Hellman problem in $\mathbb{G}$ is infeasible, and let (A, B) be a public key pair which is different from all known recipients $(A_i, B_i)_{i=1,\dots,n}$. The attacker which tries to decide whether $(A_0, B_0) = (A, B)$ has a negligible advantage (i.e. if he tries to decide whether the transaction output was sent to (A, B), then he cannot do better than guess randomly).*

Proof.

1. The signature consists of the public key $D = aR$ as well as the pair

$$h = \mathcal{H}_s\left(m \| k\mathcal{G} \| kA \| D \| R \| A\right)$$
$$t = k - rh$$

   with $k \in \mathbb{F}_\ell$ a random number. The pair $(h; t)$ is a pair of random numbers, and any pair is equally probable, which means that the attacker cannot use it to deduce any extra information. Thus the statement is equivalent to the following one: if an attacker knows R and can generate $D_i = a_i R$ for any public key $A_i = a_i\mathcal{G}$, then he cannot deduce the private key r. But if the attacker knew all private keys $\{a_i\}$, then he could trivially deduce D_i, which means that he could pick a random set of private keys $\{a_i\}$ and deduce r without any access to the signing oracle.

2. If an adversary could deduce the private transaction key with a non-negligible probability from knowing the public keys $\{A\}$ and their corresponding signatures, then he could solve the DLP in $\mathbb{G}$ for any $r \in \mathbb{F}_\ell$ and $R = r\mathcal{G}$ with a non-negligible probability by choosing random messages m_i, random private keys $a_i \in \mathbb{F}_\ell$ and public keys $A_i = a_i\mathcal{G}$, computing their signatures using the signing oracle and applying the private key deduction. Note that since the signature consists of $D = rA$ and a blinded transcript hash, this would mean that an adversary which can solve the Diffie–Hellman problem in $\mathbb{G}$ can solve the DLP in $\mathbb{G}$. Conjecturally, DHP is strictly stronger than DLP.

3. This is a stronger version of the strong soundness property, and the proof is similar. As discussed above, the signing oracle for each tuple (m, R, A) may itself depend on the random oracle, however the signature protocol shows that it only depends on the value of the random oracle at a specific point, different for each different signing oracle call. This means that we can apply the forking lemma as in strong soundness and deduce that with non-negligible probability an attacker which can forge a signature for m can compute the private transaction key. This means that either they know it, or there is a non-negligible probability (over signing oracles, random oracles and tapes) that they can solve the DLP in polynomial time, which we assume impossible.

4. This is a special case of 2 with a fixed tuple (R, A) and varying message.

5. This is a special case of 2 with a fixed public transaction key R and varying recipient public keys (and, possibly, messages).

6. To prove that (A, B) is the key pair for the output coin P_0 means to prove that for the value $D = rA = aR$ we have

$$P_0 = \mathcal{H}_s(D) + B$$

If the attacker has a non-negligible advantage in deciding this equality, then he has a non-negligible probability to produce a valid key $\mathcal{H}_s(D) + B$ given a public address (A, B), the public transaction key R and the stated data about other transaction outputs. Assuming $\mathcal{H}_s(\cdot)$ is a random oracle, a valid coin key can only be produced with non-negligible probability if the attacker can compute D with non-negligible probability. This means that he can solve the Diffie–Hellman problem for the pair (R, A), which we assume to be infeasible. Specifically, one can choose a random triple $(r, a, b) \in \mathbb{F}_\ell^3$ and set $(R, A, B) = (r\mathcal{G}, a\mathcal{G}, b\mathcal{G})$. If one can check whether P_0 is the output coin with recipient (A, B) for the public transaction key R with non-negligible advantage, then one can solve the Diffie–Hellman problem for (R, A) with non-negligible probability for arbitrary pairs (R, A).

□

6 Summary

We have introduced a protocol which allows any third party to check in a non-interactive way the validity of transaction outputs. Here we recall briefly the entire procedure:

- During the transaction, Sandra picks some of her owned coins $S_1, \ldots, S_n$ as transaction inputs. To send money to Reece and his eleven friends:
  - she generates the private transaction key r and public transaction key R,
  - and generates output coins $P_i = \mathcal{H}_s(i \| rA_{n_i}) + B_{n_i}$ for all recipient key pairs (A_k, B_k). Here i is the output coin index and n_i is the index of its recipient. She can get change by sending some amount to herself in the same transaction.
  - Finally, Sandra signs the entire transaction with the ring signature and commits it.
- If she later wants to prove that she sent money to Reece, she can generate the SendProof (D, h, t) for that transaction with Reece's public key A as described in section 4. She uses the SendProof signature to sign her private message m.

- Anyone who receives Reece's public keys (A, B), the private message m and the SendProof signature (D, h, t) can find the transaction R in the blockchain and identify all output coins in the transaction R which are received by Reece (or prove that no such outputs exist, if the signature is invalid). Technically, the key pair (A, B) is part of the signature, but it can be omitted if it is known to the verifier. This is likely to be the case with public entities like shops and financial institutions.

Overall, the SendProof signatures act as a secure and convenient way to verify payments in the Bytecoin system. Besides settling financial disputes, they can also be used as a method to check spendings for trusted wallets, provide a limited version of electronic receipts and simplify banking operations. We also believe that they will increase he trust level and convenience of using anonymous cryptocurrencies.

References

[B1] Bernstein D. J. "Curve25519: new Diffie–Hellman speed records." Pages 207–228 in *Public key cryptography—PKC 2006, 9th international conference on theory and practice in public-key cryptography, New York, NY, USA, April 24–26, 2006, proceedings,* edited by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, Tal Malkin. Lecture Notes in Computer Science 3958, Springer, **2006**. ISBN 3-540-33851-9.

[B2] "SafeCurves: choosing safe curves for elliptic-curve cryptography." https://safecurves.cr.yp.to/

[BCN] "Bytecoin: a decentralized anonymous cryptocurrency," https://bytecoin.org/
GitHub repository: https://github.com/bcndev/

[BR] Bellare M., Rogaway P. "Random oracles are practical: a paradigm for designing efficient protocols," *Proceedings of the 1st ACM conference on Computer and communications security (CCS '93).* ACM, New York, NY, USA, 62-73, **1993**.

[CNC] CryptoNote–based currencies. https://cryptonote.org/coins

[CS] Camenisch J., Stadler M. *"Proof Systems for General Statements about Discrete Logarithms,"* **1997**.

[FS] Fiat A., Shamir A. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," *Advances in Cryptology — Crypto 86 Proceedings,* Lecture Notes in Computer Science Vol. 263, A. Odlyzko ed., Springer, **1986**.

[MM] Mullen G. L., Mummert C. "Finite Fields and Applications" *Student Mathematical Library, vol. 41,* **2007**. ISBN-10: 0-8218-4212-9 ISBN-13: 978-0-8218-4418-2

[PS] Pointcheval D., Stern J. "Security Proofs for Signature Schemes," *In: Maurer U. (eds) Advances in Cryptology — EUROCRYPT '96. EUROCRYPT 1996.* Lecture Notes in Computer Science, vol 1070. Springer, Berlin, Heidelberg.

[S] van Saberhagen N. "CryptoNote v 2.0,"**2013**

[Sil] Silverman J. H. "The Arithmetic of Elliptic Curves," *Springer,* **1994**, 106.