Mini Project for Cryptography

Jinye He

Contents

1 Introduction and Definitions

Cryptography is the study of techniques for communicating securely. One of the most significant advances of modern cryptography is that modern cryptography builds up awareness of the formal definition of security. Message integrity (or message authentication) composes an important part of security. The following example will tell us the significance of message integrity.

Example 1. Imagine that a user wants to pay for a ticket to the cinema with her debit card. So the user communicates with her bank to transfer £10 from the user's account to the account of the cinema. When the bank receives this request, the bank has to consider:

1. Is the request issued by the legitimate user, or was the request issued by a malicious cheater?
2. Is this request the bank received same as the one sent by the user, or was it forged by a malicious cheater?

To achieve message integrity, the message authentication code (MAC) is proposed in the secret-key setting. Correspondingly, in a public-key setting, the digital signature scheme was proposed by Diffie and Hellman [DH76] to ensure message integrity. We formalize the definition of digital signature scheme [KL20] as follows.

Definition 2. A digital signature scheme consist of three probabilistic polynomial algorithms ($\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy}$) such that:

1. The key-generation algorithm Gen takes as input a security parameter 1^n and outputs a pair of keys (pk, sk). These are called public key and private key, respectively. We assume that each of pk and sk has a length of at least n and that n can be determined from pk or sk.
2. The signing algorithm Sign takes as input a private key sk and a message m from some message space (that may depend on pk). It outputs a signature σ, and write this as $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$.
3. The deterministic verification algorithm Vrfy takes as input a public key pk, a message m, and a signature σ. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid. We write this $b := \mathsf{Vrfy}_{pk}(m, \sigma)$.

It is required that except with negligible probability over (pk, sk) output by $\mathsf{Gen}(1^n)$, it holds that $\mathsf{Vrfy}_{pk}(m, \mathsf{Sign}_{sk}(m)) = 1$ for every legal message m.

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, $\mathcal{A}$ be an adversary, and n be the security parameter. For a fixed public key pk, the security of signature means that the adversary can't forge a valid message-signature pair in the absence of the Signer who has the private key sk. To formalize the security, we introduce the following the signature experiment [KL20].

The signature experiment $\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n)$

1. $\mathsf{Gen}(1^n)$ is run to obtain keys (pk, sk).

2. Adversary $\mathcal{A}$ is given pk and access to an oracle $\mathsf{Sign}_{sk}(\cdot)$. The adversary then outputs (m, σ). Let $\mathcal{Q}$ be the set of all queries that $\mathcal{A}$ asks its oracle.
3. $\mathcal{A}$ succeeds if and only if
   (a) $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$, and
   (b) $m \notin \mathcal{Q}$.

   In this case, the output of the experiment is defined to be 1.

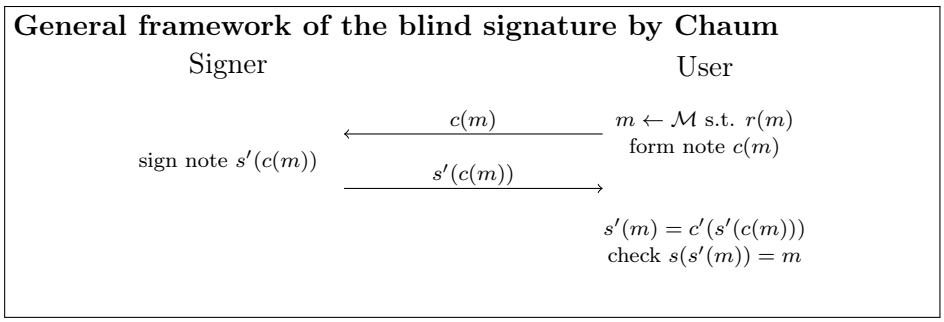
Definition 3. A signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:

$$\mathbb{P}(\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1) \leq \mathsf{negl}(n).$$

The signature scheme ensures the message integrity, while in some practical implements, we want the message to be disguised when it is signed. For example, in the circumstance of example 1, the user's payments for such as transportation, hotels, and groceries, actually enable the bank to know a lot of information about the user's private information. To protect individuals' privacy, the blind signature scheme was proposed by Chaum [Cha83] in 1983. Chaum describes his general framework of blind signature in this paper informally and we introduce it here.

Chaum's framework of blind signature The signing function s' is known only to the bank. The corresponding inverse s is public such that $s(s'(m)) = m$ and s give no clue about s'. The commuting function c and its corresponding inverse c' are both only known by the user to hide the message and yield the final signature. It is required that $s'(m) = c'(s'(c(m)))$, and $c(m)$ and s' give no clue about m. The redundancy checking predicate r checks for sufficient redundancy to make search for valid signatures impractical.

General framework of the blind signature by Chaum



Now we formalize the blind signature scheme [JLO97].

Definition 4. A blind digital signature scheme consist of two interactive parties $(\mathsf{Signer}, \mathsf{User})$ and two polynomial algorithms $(\mathsf{Gen}, \mathsf{Vrfy})$ such that:

1. The key-generation algorithm Gen is a probabilistic algorithm, taking as input a security parameter 1^n and outputs a pair of keys (pk, sk). These are called public key and private key, respectively. We assume that each of pk and sk has a length of at least n and that n can be determined from pk or sk.

2. The $\mathsf{Signer}(pk, sk)$ and $\mathsf{User}(pk, m)$ are a pair of interactive parties with probabilistic polynomial algorithms. The Signer takes as input a private key sk and the corresponding public key pk. The User takes as input the public key pk and a message m from some message space (that may depend on pk). The Signer and User engage in the interactive protocol of some polynomial (in the security parameter) number of rounds. At the end of the protocol, the Signer outputs a bit b, with $b = 1$ meaning completed and $b = 0$ meaning not-completed and the User outputs a signature $\sigma(m)$ or fail.

3. The deterministic verification algorithm Vrfy takes as input a public key pk, a message m, and a signature σ. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid. We write this $b := \mathsf{Vrfy}_{pk}(m, \sigma)$.

It is required that except with negligible probability over (pk, sk) output by $\mathsf{Gen}(1^n)$, it holds that $\mathsf{Vrfy}_{pk}(m, \sigma(m)) = 1$ for every legal message m.

Let $\Pi = (\mathsf{Gen}, (\mathsf{User}, \mathsf{Signer}), \mathsf{Vrfy})$ be a blind signature scheme, $\mathcal{A}$ be an adversary, and n be the security parameter. Similarly to the signature scheme, a secure blind signature scheme ensure that the adversary is unable to forge a valid message-signature pair. In addition to the unforgeability, security for blind signature includes blindness [JLO97] which means the signer can't pair a signed message with a particular protocol execution. Now we formalize unforgeability and blindness respectively.

The blind signature experiment $\mathsf{BSig\text{-}forge}_{\mathcal{A},\Pi}(n)$

1. $\mathsf{Gen}(1^n)$ is run to obtain keys (pk, sk).
2. Adversary $\mathcal{A}$ controls the 'User' and is given pk. $\mathcal{A}(pk)$ engages in polynomially many (in n) adaptive, parallel and arbitrarily interleaved interactive protocols with Signer. Then, $\mathcal{A}$ outputs a (m, σ). Let $\mathcal{Q}$ be the set of messages and corresponding signatures gotten from interactive protocols.
3. $\mathcal{A}$ succeeds if and only if
   (a) $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$, and
   (b) $m \notin \mathcal{Q}$.

   In this case, the output of the experiment is defined to be 1.

Definition 5. A blind signature scheme $\Pi = (\mathsf{Gen}, (\mathsf{User}, \mathsf{Signer}), \mathsf{Vrfy})$ is unforgeable, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:

$$\mathbb{P}(\mathsf{BSig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1) \leq \mathsf{negl}(n).$$

Similarly, we formalize blindness for blind signature schemes.

The blind signature experiment $\mathsf{BSig\text{-}crp}_{\mathcal{A},\Pi}(n)$

1. $\mathsf{Gen}(1^n)$ is run to obtain keys (pk, sk).
2. Adversary $\mathcal{A}$ controls the 'Signer' and produces two messages $\{m_0, m_1\}$, polynomial in n, where $\{m_0, m_1\}$ are lexicographically ordered and may even depend on pk and sk.
3. We denote by $\{m_b, m_{1-b}\}$ the same two messages $\{m_0, m_1\}$, ordered according to the value of uniform bit $b \in \{0, 1\}$, where the value of b is hidden from $\mathcal{A}$. $\mathcal{A}(n, pk, sk, m_0, m_1)$ engages in two parallel interactive protocols, the first with $User(pk, m_b)$ and the second with $User(pk, m_{1-b})$.
4. After the user outputs the signatures $\{\sigma(m_b), \sigma(m_{1-b})\}$. $\mathcal{A}$ is given as an additional input $\{\sigma(m_b), \sigma(m_{1-b})\}$ ordered according to the corresponding $\{m_0, m_1\}$ order.
5. Adversary $\mathcal{A}$ outputs a bit $\tilde{b}$. The output of the experiment is 1 if $\tilde{b} = b$, and 0 otherwise. If $\tilde{b} = b$ we say that $\mathcal{A}$ succeeds.

Definition 6. A blind signature scheme $\Pi = (\mathsf{Gen}, (\mathsf{User}, \mathsf{Signer}), \mathsf{Vrfy})$ is $\mathsf{computational\ blind}$, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:

$$\mathbb{P}(\mathsf{BSig\text{-}crp}_{\mathcal{A},\Pi}(n) = 1) \leq \frac{1}{2} + \mathsf{negl}(n).$$

Overview In the next two sections, we will introduce two simple blind signature schemes, the blind modification of DSA (digital signature algorithm) and the blind Schnorr signature scheme. These two schemes are both based on the discrete logarithm problem and before the introduction of each, we will first state the corresponding not-blind scheme. In addition, for the blind Schnorr signature, we give an implement in SAGE. In section 4, we will discuss one of the most important applications of blind signature, electronic cash (e-cash). As a currency, e-cash requires more security property than just blindness and unforgeability. We will mainly discuss the property against double-spending. At the end of this section, we will compare e-cash with cryptocurrency (take Bitcoin as an example).

2 Blind signature scheme based on a modification of DSA

2.1 A modification of DSA

The digital signature algorithm (DSA) is a standardized digital signature scheme. Here we introduce a modification of DSA (MDSA) proposed by Camenisch, Piveteau, and Stadler [CPS95]. The scheme works in a q-order cyclic subgroup of $\mathbb{Z}_p^*$ where p, q are primes such that $q|(p-1)$. The bit length

of q is the security parameter. After selecting proper primes p, q, find a generator g of an order-q subgroup of $\mathbb{Z}_p^*$. These parameters are called domain parameter. Now we introduce the MDSA scheme.

A modification of DSA (MDSA)
Let $\mathcal{G}$ be the domain-parameter-generation algorithm.

1. Gen: on input 1^n, run $\mathcal{G}(1^n)$ to obtain (p, q, g). Choose uniform $x \in \mathbb{Z}_q$ and set $y := g^x \mod p$. The public key is (p, q, g, y) and the private key is x.
2. Sign: on the input the private key x and a message m which is an integer relatively prime to q. Choose uniform $k \in \mathbb{Z}_q$, and compute R, r and s given by:

$$\begin{aligned} R &= g^k && \mod p, \\ r &= R && \mod q, \\ s &= km + rx && \mod q. \end{aligned}$$

   Output the signature (r, s).
3. Vrfy: on the input a public key (p, q, g, y), a message m, and a signature (r, s). Compute

$$T = (g^s y^{-r})^{m^{-1}} * \mod p.$$

   Output 1 if and only if

$$r = T \mod q$$

   and 0 otherwise.

2.2 The blind MDSA

Now we introduce the blind signature scheme based on the MDSA given in [CPS95]. The blind MDSA (BMDSA) works also in a q-order cyclic subgroup of $\mathbb{Z}_p^*$ where p, q are primes such that $q|(p-1)$. The setting of the domain parameter is the same as the one in MDSA.

The blind MDSA (BMDSA)
Let $\mathcal{G}$ be the domain-parameter-generation algorithm.

1. Gen: on input 1^n, run $\mathcal{G}$ to obtain (p, q, g). Choose uniform $x \in \mathbb{Z}_q$ and set $y := g^x$. The public key is (p, q, g, y) and the private key is x.
2. Interactive protocol:
   (a) i. On the input the private key x and public key (p, q, g, y), the Signer randomly chooses $\tilde{k} \in \mathbb{Z}_q$ and compute

$$\tilde{R} = g^{\tilde{k}} \mod p.$$

*m^{-1} denotes the inverse of m modulo q

ii. Check whether $gcd(\tilde{R}, q) = 1$. If this is not the case, the Signer goes back to the last step. Otherwise, send $\tilde{R}$ to the User.

(b) i. The User check whether $gcd(\tilde{R}, q) = 1$. If this is not the case, the User outputs fail and stops the protocol. Correspondingly, the Signer outputs 0 meaning not-completed.

ii. The User randomly chooses $\alpha,\ \beta \in \mathbb{Z}_q$ and computes

$$R = \tilde{R}^{\alpha} g^{\beta} \mod p.$$

iii. Check whether $gcd(R, q) = 1$. If this is not the case, the User goes back to the last step. Otherwise, compute

$$\tilde{m} = \alpha m \tilde{R} R^{-1} \mod q$$

where m is an integer relatively prime to q and send $\tilde{m}$ to the Signer.

(c) The Signer computes $\tilde{s} = \tilde{k}\tilde{m} + \tilde{R}x \mod q$ and sends $\tilde{s}$ to User. Output 1 meaning completed.

(d) The User computes $s = \tilde{s}R\tilde{R}^{-1} + \beta m \mod q$ and $r = R \mod q$. Output (r, s) as a signature.

3. Vrfy: on the input a public key (p, q, g, y), a message m, and a signature (r, s). Compute

$$T = (g^s y^{-r})^{m^{-1}} \mod p.$$

Output 1 if and only if

$$r = T \mod q$$

and 0 otherwise.

2.3 Security analysis

Since these schemes are based on the discrete logarithm problem, one might expect they are unforgeable. Unfortunately, the adversary can forge a signature on an arbitrary message since the signatures of m and $m + q$ are the same.

In terms of blindness, the BMDSA has a stronger property than computational blindness. It is statistical blindness which means Signer's complete view of execution of the protocol and the message-signature pair $(m, \sigma(m))$ are statistically independent. Let $\mathcal{V}$ be the space of Signer's complete view of the execution of the protocol. Let $\mathcal{M}, \mathcal{S}$ be the message space and signature space. Let V, M, S be the random variable denoting the view, message, and signature respectively. Now we can give a formalized definition of statistical blindness.

Definition 7. A blind signature scheme $\Pi = (\mathsf{Gen}, (\mathsf{User}, \mathsf{Signer}), \mathsf{Vrfy})$ is statistical blind, if for every probability distribution over V, and (M, S), any view $v \in \mathcal{V}$ and valid message-signature pair $(m, \sigma(m)) \in (\mathcal{M}, \mathcal{S})$ such that:

$$\mathbb{P}(V = v | (M, S) = (m, \sigma(m))) = \mathbb{P}(V = v).$$

Now we prove the following theorem.

Theorem 8. BMDSA *is statistical blind.*

Proof. Recall the BMDSA protocol. If the message-signature pair $(m,(r,s))$ has been generated during an execution of the protocol with view v consisting of $\tilde{k}$, $\tilde{R} = g^{\tilde{k}} \mod p$, $\tilde{m}$, and $\tilde{s}$, then the following equations must hold:

$$\begin{aligned} \tilde{m} &= \alpha m \tilde{R} r^{-1} && \mod q \\ s &= \tilde{s} r \tilde{R}^{-1} + \beta m && \mod q \\ r &= \tilde{R}^{\alpha} g^{\beta} && \mod p \mod q. \end{aligned}$$

Since $m, \tilde{R}$, and r are relatively prime to q, we can uniquely determined by the first two equations:

$$\begin{aligned} \alpha &= \tilde{m} m^{-1} r \tilde{R}^{-1} && \mod q \\ \beta &= (s - \tilde{s} r \tilde{R}^{-1}) m^{-1} && \mod q. \end{aligned}$$

By substituting $\tilde{s} = \tilde{k}\tilde{m} + \tilde{R}x \mod q$, we can easily check that the α and β got above fit the whole protocol:

$$\begin{aligned} \tilde{k}\alpha + \beta = \tilde{k}\tilde{m} m^{-1} r \tilde{R}^{-1} + s m^{-1} - \tilde{s} r \tilde{R}^{-1} m^{-1} = (s - rx) m^{-1} && \mod q \\ \tilde{R}^{\alpha} g^{\beta} = g^{\tilde{k}\alpha + \beta} = g^{(s-rx)m^{-1}} = (g^s y^{-r})^{m^{-1}} = T && \mod p \end{aligned}$$

and $r = T \mod q$. Therefore, we have

$$\mathbb{P}(V = v | (M,S) = (m,(r,s))) = \mathbb{P}(A = \alpha, B = \beta) = \frac{1}{q^2}.$$

In terms of $\mathbb{P}(V = v)$, it is easy to check that $\tilde{k}$ is randomly chosen from $\mathbb{Z}_q$, $\tilde{R}$ is determined by $\tilde{k}$. Since α is randomly chosen from $\mathbb{Z}_q$ and $m, \tilde{R}$, R are relatively prime to q, $\tilde{m}$ is uniform from $\mathbb{Z}_q$. Also, $\tilde{s}$ is determined by $\tilde{k}$ and $\tilde{m}$ since x is determined in advance. Thus we have

$$\mathbb{P}(V = v) = \frac{1}{q^2}$$

which implies that

$$\mathbb{P}(V = v | (M,S) = (m, \sigma(m))) = \mathbb{P}(V = v).$$

□

Statistical blindness is stronger than computational blindness, and we have the following proposition.

Proposition 9. *Let* $\Pi = (\mathsf{Gen}, (\mathsf{User}, \mathsf{Signer}), \mathsf{Vrfy})$ *is a blind signature. If* Π *is statistical blind, then it is computational blind.*

Proof. Recall the blind signature experiment $\mathsf{BSig\text{-}crp}_{\mathcal{A},\Pi}(n)$. The adversary $\mathcal{A}$ succeeds if and only if $b = \tilde{b}$. Since b is chosen uniformly from $\{0,1\}$, we have

$$\begin{aligned}\mathbb{P}(\mathcal{A}\text{ succeeds}) &= \mathbb{P}(\tilde{b}=0|b=0)\mathbb{P}(b=0) + \mathbb{P}(\tilde{b}=1|b=1)\mathbb{P}(b=1)\\ &= \frac{1}{2}\Big(\mathbb{P}(\tilde{b}=0|b=0) + \mathbb{P}(\tilde{b}=1|b=1)\Big)\\ &= \frac{1}{2}\Big(\mathbb{P}(\tilde{b}=0) + \mathbb{P}(\tilde{b}=1)\Big)\\ &= \frac{1}{2} \leq \frac{1}{2} + \mathsf{negl}(n)\end{aligned}$$

where the third equation comes from statistical blindness.

□

By this proposition, we have the following result.

Theorem 10. BMDSA *is computational blind.*

3 Blind signature scheme based on Schnorr signature

3.1 Schnorr signature

Another simple and well-known signature based on the hardness of the discrete logarithm problem is the Schnorr signature. The Schnorr signature and corresponding blind Schnorr introduced here refer to [Sch01]. The Schnorr signature also works in a q-order cyclic subgroup of $\mathbb{Z}_p^*$ where p, q are primes such that $q|(p-1)$. We use the same algorithm as the one in the MDSA to get domain parameters. Now we introduce the Schnorr signature scheme.

Schnorr signature scheme

Let $\mathcal{G}$ be the domain-parameter-generation algorithm.

1. Gen: on input 1^n, run $\mathcal{G}$ to obtain (p,q,g). Choose uniform $x \in \mathbb{Z}_q$ and set $y := g^x \mod p$. The public key is (p,q,g,y) and the private key is x.
2. Sign: on the input the private key x and a message $m \in \{0,1\}^*$. Choose uniform $k \in \mathbb{Z}_q$, and computes r, e and s given by:

$$\begin{aligned} r &= g^k && \mod p\\ e &= H(m,r) && \mod q\\ s &= k + ex && \mod q\end{aligned}$$

   where H is a hash function. Output the signature (e,s).
3. Vrfy: on the input a public key (p,q,g,y), a message m, and a signature (e,s). Compute $r = g^s y^{-e} \mod p$ and output 1 if and only if

$$e = H(m,r) \mod q$$

   and 0 otherwise.

3.2 The blind Schnorr signature

The blind Schnorr signature based on the Schnorr signature also has the same setting of domain parameters.

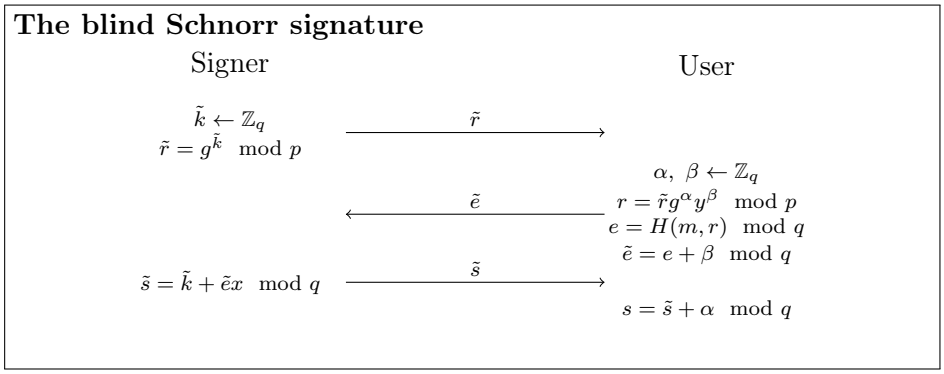
Blind Schnorr signature scheme

Let $\mathcal{G}$ be the domain-parameter-generation algorithm.

1. Gen: on input 1^n, run $\mathcal{G}$ to obtain (p, q, g). Choose uniform $x \in \mathbb{Z}_q$ and set $y := g^x \mod p$. The public key is (p, q, g, y) and the private key is x.
2. Interactive protocol:
   (a) On the input the private key x, the Signer randomly chooses $\tilde{k} \in \mathbb{Z}_q$ and compute
   $$\tilde{r} = g^{\tilde{k}} \mod p.$$
   Send $\tilde{r}$ to the User.
   (b) On the input the message m and the public key (p, q, g, y), the User randomly chooses $\alpha,\ \beta \in \mathbb{Z}_q$ and computes
   $$\begin{aligned} r &= \tilde{r} g^{\alpha} y^{\beta} && \mod p \\ e &= H(m, r) && \mod q \\ \tilde{e} &= e + \beta && \mod q \end{aligned}$$
   and sends $\tilde{e}$ to the Signer.
   (c) The Signer computes $\tilde{s} = \tilde{k} + \tilde{e}x \mod q$ and sends $\tilde{s}$ to User. Output 1 meaning completed.
   (d) The User computes $s = \tilde{s} + \alpha \mod q$. Output (r, e, s) as a signature.
3. Vrfy: on the input a public key (p, q, g, y), a message m, and a signature (e, s). Compute $r = g^s y^{-e} \mod p$ and output 1 if and only if
   $$e = H(m, r) \mod q$$
   and 0 otherwise.

Compare the blind Schnorr signature with Chaum's framework. We notice that there is an initialization step (the signer chooses a $\tilde{k}$ randomly and sends $g^{\tilde{k}}$ to the User) in blind Schnorr's signature. This step is designed to keep the private key x secure, without which, it is easy to compute x by $\tilde{e}$ and $\tilde{s}$.

Except for this step, the blind Schnorr signature fits the framework proposed by Chaum roughly. In terms of details, there are some differences. First, in Chaum's framework, the message m is available from the stripped signature $s'(m)$, while in the blind Schnorr signature, the message should be provided in addition to the signature. The second difference is that in the blind Schnorr signature, the signer only blindly signs part of the signature. The final signature contains not only the stripped signature $c'(s'(c(m)))$, i.e. s in the blind Schnorr signature, but also e. In addition, due to the application of Hash function, we omit the redundancy checking function r.



3.3 Security analysis

3.3.1 Unforgeability

In this section, we will discuss the unforgeability of the Schnorr signature and blind Schnorr signature. It is difficult to give security proofs in the standard model in which the adversary is only limited by the amount of time and computational power available. Security proofs are given based on more idealized models. For example, the random-oracle model (ROM) idealizes the Hash function in cryptographic schemes. The belief [KL20] is:

A proof of security in the random-oracle model is significantly better than no proof at all.

In terms of the group, there are two idealized models, the generic group model (GGM) and the algebraic group model (AGM). The Generic Group Model (GGM) proposed by Shoup [Sho97] formalizes the idea that group elements don't give any information about the structure of the group and the adversary only has access to a simple addition oracle. The algebraic group model (AGM) proposed by Fuchsbauer, Kiltz, and Loss [FKL18] lies between GGM and the standard model. In the algebraic group model (AGM), an adversary gets as input the group elements $(g_1, g_2, \cdots, g_n)$ and outputs X, it must also output the vector $\vec{\alpha}$ such that $X = \alpha_0 \prod_{i=1}^{n} g_i^{\alpha_i}$.

Schnorr signature The Schnorr signature is proved secure based on the discrete logarithm assumption in the combination of AGM and ROM by Fuchsbauer, Plouviez, and Seurin [FPS20]. Another negative result is that the security of the Schnorr signature cannot be reduced to the discrete logarithm problem [PV05] if one-more discrete logarithm problem is assumed hard in the standard model. The one-more discrete logarithm problem enables the adversary to access a discrete logarithm oracle and requires the adversary to compute one more discrete logarithm problem than the number of oracle calls it made. It is easy to check that the one-more discrete logarithm problem is simpler than the discrete logarithm problem.

Blind Schnorr signature One might expect the security of the blind Schnorr signature to be the same as the Schnorr signature. However, the blind Schnorr signature is not secure based on the discrete logarithm assumption even in GGM and ROM. In 2001, Schnorr propose that the ROS (Random inhomogeneities in a Overdetermined Solvable system of linear equations) problem should also be hard to ensure the security of the blind signature. The security proof in the AGM+ROM

model is given by Plouviez [Plo21] assuming that both the one-more discrete logarithm problem and the ROS problem are hard.

Unfortunately, an algorithm has been proposed by Benhamouda, Lepoint, Loss, Orrù, and Raykova [BLL+22] to solve the ROS problem $\bmod p$ in polynomial time for $\ell > \log p$ dimensions. If concurrent executions are allowed, this algorithm leads to practical attacks against the blind signature scheme. Before this work, Wagner [Wag02] propose a sub-exponential attack against the ROS problem. A modification of blind Schnorr proposed by Fuchsbauer, Plouviez, and Seurin [FPS20] to resist Wagner's attack also can't be broken by Benhamouda et al.'s attack.

3.3.2 Blindness

In terms of the blindness of the blind Schnorr signature scheme, we will show that this scheme is also statistical blind.

Theorem 11. *The blind Schnorr signature scheme is statistical blind.*

Proof. Recall the blind Schnorr signature protocol. If the message-signature pair $(m, (e, s))$ has been generated during an execution of the protocol with view v consisting of $\tilde{k}$, $\tilde{r} = g^{\tilde{k}} \mod p$, $\tilde{e}$, and $\tilde{s}$, then we can easily find that α, β are uniquely determined by:

$$\begin{aligned} \alpha &= s - \tilde{s} \mod q \\ \beta &= \tilde{e} - e \mod q. \end{aligned}$$

It is easy to check that the α and β got above fit the whole protocol. Therefore, we have

$$\mathbb{P}(V = v | (M, S) = (m, (r, s))) = \mathbb{P}(A = \alpha, B = \beta) = \frac{1}{q^2}.$$

In terms of $\mathbb{P}(V = v)$, since $\tilde{k}$ and β are randomly chosen from $\mathbb{Z}_q$, $\tilde{e}$ and $\tilde{s}$ are uniformly distributed in $\mathbb{Z}_q$ independently. $\tilde{r}$ is determined by

$$\tilde{r} = g^{\tilde{k}} = g^{\tilde{s}} y^{-\tilde{e}} \mod p.$$

Thus we have

$$\mathbb{P}(V = v) = \frac{1}{q^2}$$

which implies that

$$\mathbb{P}(V = v | (M, S) = (m, \sigma(m))) = \mathbb{P}(V = v).$$

□

By proposition 9, we have the following result.

Theorem 12. *The blind Schnorr signature scheme is computational blind.*

Zero-knowledage protocol From the proof of theorem 11, we noticed that the transcripts of the blind Schnorr signature protocol $(\tilde{r}, \tilde{e}, \tilde{s})$ have the following properties:

- $\tilde{e}$ and $\tilde{s}$ are uniform in $\mathbb{Z}_q$ and
- $\tilde{r}$ is determined by $\tilde{r} = g^{\tilde{k}} = g^{\tilde{s}} y^{-\tilde{e}} \mod p$.

Therefore there is a simulator S given $y = g^x \mod p$ that can just sample $\tilde{e}', \tilde{s}' \leftarrow \mathbb{Z}_p$ and compute $\tilde{r}'$ to generate a tuple $(\tilde{r}', \tilde{e}', \tilde{s}')$ with the same distribution with real signature protocol which means the blind Schnorr signature is a Zero-knowledge protocol.

3.4 Implementation of the blind Schnorr signature

We have stated the blind Schnorr signature scheme and in this subsection, we show a simple implementation in SAGE. There are four files gen.sage, signer.sage, user.sage, and vrfy.sage serving as the four components in the blind signature scheme. In the blind_schnorr.sage, we call functions in these four files to generate a key pair, complete the interactive protocol, and verify the message-signature pair. We will introduce the four files one by one and the main file at last.

gen.sage We just use the Crypto.PublicKey package in the PyCryptodome to generate a key pair. There are different key types available and we use the DSA keys[Leg]. The function DSA.generate() takes one of 1024, 2048 and 3072 as input and outputs a key object including the following variables.

- p (integer)—DSA modulus
- q (integer)—Order of the subgroup
- g (integer)—Generator
- x (integer)—Private key
- y (integer)—Public key

The three inputs correspond to three choices of L and N (the bit lengths of p and q) as follows. In the code, we take 2048 as an example.

1. L=1024 and N=160
2. L=2048 and N=224
3. L=3072 and N=256

Now we list the code of gen.sage. We multiply the key variable with 1 to change its type to sage integer and use IntegerModRing() to change the variable's type to integer in finite rings.

```
from Crypto.PublicKey import DSA

#call function of DSA to generate a key pair
def create_key(n):
```

```
    key=DSA.generate(int(n))
    p=1*key.p
    q=1*key.q
    Rp=IntegerModRing(p)
    g=Rp(key.g)
    Rq=IntegerModRing(q)
    x=Rq(key.x)
    y=Rp(key.y)
    return(p,q,g,x,y)
```

signer.sage We use global variables to save information. The function assgin_key_signer() receives the keys and saves them. The function init_sig() randomly choose $\tilde{k}$ and return $\tilde{r}$ for User. Finally the function bsig() completes the blind signature.

```
import secrets

#initialize global variables
p=1
q=1
g=1
x=1
k=1

#obtain keys and save them in global variables
def assgin_key_signer(a,b,c,d):
    global p,q,g,x
    p=a
    q=b
    g=c
    x=d


#initilize the signature protocol
def init_sig():
    global k,q,p,g
    k=secrets.randbelow(q)
    R=IntegerModRing(p)
    r=R(g)^k
    return(r)

#blindly sign the message received from the user
def bsig(e):
    global k,x,q
    R=IntegerModRing(q)
    s=R(k+e*x)
    return(s)
```

user.sage We use global variables to save information. The function assgin_key_user() receives the keys and saves them. The function blind() blind the message and return $\tilde{s}$ for Signer to sign. Finally the function sig() yields the final signature.

```
import secrets
import hashlib

#initialize global variables
p=1
q=1
g=1
y=1
a=1
e=1
r=1

#obtain keys and save them in global variables
def assgin_key_user(a,b,c,d):
    global p,q,g,y
    p=a
    q=b
    g=c
    y=d

#blind the message and return the covered message for signer
def blind(rr,m):
    global p,q,g,y,a,e,r
    a=secrets.randbelow(q)
    Rq=IntegerModRing(q)
    a=Rq(a)
    b=secrets.randbelow(q)
    b=Rq(b)
    Rp=IntegerModRing(p)
    r=Rp(rr*(Rp(g)^(a))*(Rp(y)^(b)))
    h=str(m)+str(r)
    e=Rq(int.from_bytes(hashlib.sha256(h.encode()).digest()))
    ee=Rq(e+b)
    return(ee)

#yield the signature
def sig(ss):
    global p,q,g,y,a,e,r
    Rq=IntegerModRing(q)
    s=Rq(ss+a)
    return(e,s)
```

vrfy.sage We use global variables to save information. The function assgin_key_vrfy() receives the keys and saves them. The function vrfy() takes a message-signature pair as input and output a

bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid.

```
import hashlib

#initialize global variables
p=1
q=1
g=1
y=1

#obtain keys and save them in global variables
def assgin_key_vrfy(a,b,c,d):
    global p,q,g,y
    p=a
    q=b
    g=c
    y=d

#verify whether a message-signature pair is valid or not
def vrfy(m,ee,ss):
    Rp=IntegerModRing(p)
    rrch=(Rp(g)^ss)*(Rp(y)^(-ee))%p
    h=str(m)+str(rrch)
    Rq=IntegerModRing(q)
    ch=Rq(int.from_bytes(hashlib.sha256(h.encode()).digest()))
    if ch==ee:
        return 1
    else:
        return 0
```

blind_schnorr.sage In the main file, we first call the create_key() function in gen.sage to generate a proper key pair and then assign keys to Signer, User, and Verifier. After that, a message is randomly chosen from $\mathbb{Z}_p^*$ (this setting is just for simplification) and an interactive protocol is executed to obtain its signature. Finally, we call the function in vrfy.sage to verify the message-signature pair and print the result.

```
import secrets
load("gen.sage")
load("signer.sage")
load("user.sage")
load("vrfy.sage")

#set security parameter
n=2048

#generate key pair
key=create_key(n)
p=key[0]
```

```
q=key[1]
g=key[2]
x=key[3]
y=key[4]

#assign keys
assgin_key_signer(p,q,g,x)
assgin_key_user(p,q,g,y)
assgin_key_vrfy(p,q,g,y)

#randomly choose a message
m=secrets.randbelow(p)

#ineractive protocol to signature blindly
rr=init_sig()
ee=blind(rr,m)
ss=bsig(ee)
signature=sig(ss)

e=signature[0]
s=signature[1]

#verify and print the result
b=vrfy(m,e,s)
print("message:",m)
print("signature:",signature)
if b==1:
    print('valid');
else:
    print('invalid')
```

3.4.1 Parameter and hash function choice

Federal Information Processing Standards (FIPS) [Sta23] and Recommendations of key management (SP 800-57 Part 1 Rev. 5) [BD20] are published by the National Institute of Standards and Technology (NIST) to specify cryptographic techniques for protecting sensitive, unclassified information. We choose the parameters according to the recommendations given in these two documents.

First, we notice that the security of the blind Schnorr signature is ensured by the assumption that the discrete logarithm problem is hard and the hash function used in the protocol is difficult to inverse. The security strength of the protocol is determined by the weaker security strength among the discrete logarithm problem and inverse hash function. Since the discrete logarithm assumption, the security strength relates to the N (the bit length of q i.e. security parameter of the blind Schnorr signature), and the corresponding L is the bit length of p. Meanwhile, different hash functions have different security strengths. We list the security strength of different (L, N) choices and different hash function[†] given by [BD20]. FIPS specifies the following choices for the pair L and N.

[†]Hash functions list here are Secure Hash Functions which are a family of cryptographic hash functions published

Security Strength	FFC (finite-field cryptography)
≤ 80	L=1024, N=160
112	L=2048, N=224
128	L=3076, N=256
192	L=7680, N=384
256	L=15360, N=512

Table 1: Security strengths of different (L, N)

Security Strength	hash function
≤ 80	SHA-1
112	SHA-224, SHA-512/224, SHA3-224
128	SHA-256, SHA-512/256, SHA3-256
192	SHA-384, SHA3-384
256	SHA-512, SHA3-512

Table 2: Security strengths of different hash function

- L=1024 and N=160
- L=2048 and N=224
- L=2048 and N=224
- L=3072 and N=256

It is recommended that a Federal Government entity other than a Certification Authority (CA) should[‡] use only the first three (L, N) pairs. A CA shall[§] use an (L, N) pair that is equal to or greater than the (L, N) pairs used by its subscribers. In our implementation, we choose $(2048, 224)$ and SHA-256 to ensure security[¶].

3.4.2 Efficiency analysis

In this part, we try to analyze the efficiency of our implementation of the blind Schnorr signature.

Running time We import the package time and use the function time() (return the time in seconds as a floating point number) to get the time. Since the key pair and the message can be prepared in advance, the running time of the signature only contains the interactive protocol (step 2 in the blind Schnorr signature scheme). We record the start time and the end time. The difference

by the NIST as a FIPS.

[‡]'Should' means a strong recommendation, but not a requirement of FIPS.

[§]'Shall' means a requirement of this Standard.

[¶]Since the algorithm proposed by Benhamouda et al., the blind Schnorr signature is insecure if concurrent executions are allowed. We rule this attack in this Implementation out.

Number of runs	Running time	Average running time
10	0.021070268154144287	0.0021070268154144287
100	0.22790222644805908	0.0022790222644805908
1000	2.285622470378876	0.002285622470378876

Table 3: Running time of the blind Schnorr signature

between the two is the running time. Since the running time of single execution of the protocol is uncertain, we always run the protocol a number of times and compute the average running time. Here we list several data in table 3.

Asymptotic complexity We count the number of operations and number of times that subroutines are called and list the number and corresponding computational complexity in the table 4. Since the complexity of different multiplication algorithms is different (from $O(N \log N)$ to $O(N^2)$ referring to [Wika] for details), we use $M(*)$ below denotes the complexity of the chosen multiplication algorithm. Therefore the total complexity of the signature protocol is $O(M(L)N)$.

Operation/subroutine	Bit-length of input	Complexity	Number of times
Addition	N	$\Theta(N)$	3
Multiplication	N	$M(N)$	1
	L	$M(L)$	2
Modular exponentiation	L,N	$O(M(L)N)$	3
secrets.randbelow()	N	$\Theta(N)$	3
sha256()	L+N	$O(L+N)$	1

Table 4: Number of operations and times that functions are called

4 Electronic cash

As we mentioned above, untraceable electronic cash (e-cash) is the motivation for Chaum to propose the blind signature scheme. Nowadays, blind signature schemes have been used in various applications, such as e-cash, e-voting, and anonymous credentials, to protect users' privacy. In this section, we will discuss the e-cash system [Raz02, HLMN06, DKL15] mainly.

4.1 Security properties

Security-related properties As a currency, the first property that e-cash must guarantee is that e-cash can't be forged maliciously and can be verified its integrity by shops and banks. In addition, the same paper notes or coins can't be spent more than once in real life. E-cash systems should also resist double-spending.

- unforgeability

- authentication

- double-spending identification

Privacy-related properties In real life, banks are unable to trace the paper notes withdrawn from them or link two payments to the same user by paper notes. These untraceability and unlinkability protect consumer's privacy from financial institutions. E-cash should also ensure such anonymity. Meanwhile, the possession of paper notes or coins are kept secret from others, e-cash should also be confidential.

- untraceability

- unlinkability

- confidentiality

To protect privacy while ensuring security, Chaum proposes two notions: 'electronic coins' and 'blind signature'. The former means every 'coin' (signed message) is worth a fixed amount and the latter means that the bank can't pair the signed message with an exact protocol execution. However, Chaum's approach can't resist double-spending in off-line circumstances. In the next section, we will focus on the property—double-spending identification.

4.2 Double-spending

The e-cash system always involves three parties User with an account at the bank, Bank which authorizes the electronic coins, and Seller who accepts electronic coins. Each e-cash protocol always contains three phases as follows.

1. **Withdrawal**: the User withdraws an electronic coin from the Bank which means the Bank give a signature of a electronic coin and debits the User's account.

2. **Payment**: the User spends the coin by executing a transaction with a Seller which means the User should prove his possession of the coin and generating the payment transcript.

3. **Deposit**: the Seller deposits the transaction at the Bank, which credits the Seller's account.

In terms of the circumstance of the transaction, if there is continuous communication between the Seller and Bank, the Bank can detect whether the electronic coin is spent twice by its database immediately. We say such circumstance is 'on-line'. The Seller can reject to accept the coin and resist double-spending. Chaum's approach can solve the on-line double-spending problem. Unfortunately, in an 'off-line' context without continuous communication between the Seller and Bank, it is too late when the Bank detect the double-spending and can't identify who makes the cheat.

The approaches to solving the off-line double-spending problem are unsatisfying before 1994. They require either a great sacrifice in efficiency or seem to have questionable security, if not both. In 1994, Brands [Bra94] proposes his untraceable off-line cash system to solve this problem. This system even offers a prior restraint of double-spending by introducing an 'observer' in the

wallet. Now we first define the double-spending identification [DKL15] and then introduce Brands' approach.

Let ID be the set of User's identities. Let TR be the set of all transactions and there is a function $transID : TR \to TRS \times C$ where TRS is the set of all payment transcripts and C is the set of all coins. The function $transID$ will map any transaction $tr \in TR$ to its transcripts $trs \in TRS$ which can identify tr and the coin $c \in C$ involved in this transaction.

Definition 13. An e-cash system is double-spending identifiable if there exists a test $T_{DSI} : TR \times TR \to ID$ satisfying that for any two different valid transactions tr_1 and tr_2 that are involve the same coin (i.e., $transId(tr_1) = (trs_1, c)$ and $transId(tr_2) = (trs_2, c)$ such that $trs_1 \neq trs_2$), $T_{DSI}(transId(tr_1), transId(tr_2))$ will output the ID of the User who withdraw the coin c.

Now we introduce the Brands' system simply to explain why it is double-spending identifiable. Brands' system works in a q-order cyclic group G with a generator-tuple (g, g_1, g_2) which is public. The secret key $x \leftarrow \mathbb{Z}_q^*$ is kept secret by the Bank. The public key is $(g, g_1, g_2, h, y_1, y_2)$ where $h = g^x, y_1 = g_1^x, y_2 = g_2^x$.

Opening an account The User randomly choose a number $u \in \mathbb{Z}_q$ and sends $I = g_1^u$ to the Bank and the Bank will identify the User by I.

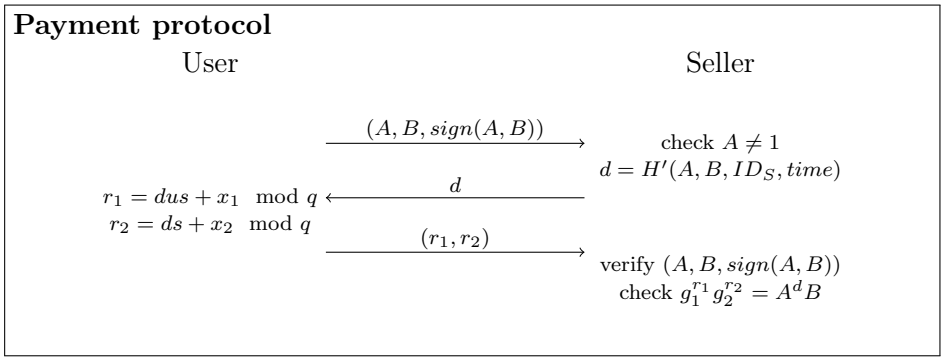
Withdrawal protocol When the User wants to withdraw a coin at Bank, the following protocol is performed where H is a hash function. A coin withdrawn by this protocol is a triple $(A, B, sign(A, B))$ where $sign(A, B) = (z, a, b, r)$ such that

$$g^r = h^{H(A,B,z,a,b)}a \quad \text{and} \quad A^r = z^{H(A,B,z,a,b)}b.$$

Withdrawal protocol

Bank(Signer)		User
$\tilde{w} \leftarrow \mathbb{Z}_q$ $\tilde{a} = g^{\tilde{w}}$ $\tilde{b} = (Ig_2)^{\tilde{w}}$	$\xrightarrow{\tilde{a}, \tilde{b}}$	
		$s \leftarrow \mathbb{Z}_q^*$ $x_1, x_2, \alpha, \beta \leftarrow \mathbb{Z}_q$ $A = (Ig_2)^s$ $B = g_1^{x_1} g_2^{x_2}$ $z = (y_1^u y_2)^s$ $a = \tilde{a}^\alpha g^\beta$ $b = \tilde{b}^{s\alpha} A^\beta$ $c = H(A, B, z, a, b)$ $\tilde{c} = c/\alpha \mod q$
	$\xleftarrow{\tilde{c}}$	
$\tilde{r} = \tilde{w} + \tilde{c}x \mod q$	$\xrightarrow{\tilde{r}}$	
		check $g^{\tilde{r}} = h^{\tilde{c}}\tilde{a}$ check $(Ig_2)^{\tilde{r}} = (y_1^u y_2)^{\tilde{c}}\tilde{b}$ $r = \tilde{r}\alpha + \beta \mod q$

Payment protocol When the User wants to spend his coin at Seller S, the following protocol is performed where ID_S is the identity of the Seller and H' is a hash function.



When the same coin is spent twice which means that there are different transcripts (d, r_1, r_2) and (d', r_1', r_2') for one coin, it is easy to get

$$g_1^{(r_1 - r_1')/(r_2 - r_2')} = g_1^u = I.$$

Therefore, we can identify who double-spends the coin. Brands also propose the notion of 'wallet with observers' in which the User can spend the coin only with the cooperation of the observer. The observer will delete the information of the spent coin which ensure prior restraint of double-spending. We refer interested readers to [Bra94].

4.3 Electronic cash v.s. cryptocurrency

In this section, we will compare e-cash (take Chaum's system [Cha83] as the example) and cryptocurrency (take Bitcoin [Nak08] as the example) in terms of anonymity, trust assumptions and efficiency as well as the societal implication of these differences.

4.3.1 Anonymity

In Chaum's system, the blind signature scheme is used to ensure anonymity. During the withdrawal, the bank gives its certificate without knowing the electronic coin and is also unable to link a particular execution of the blind signature to an authenticated coin. Under exceptional circumstances, Chaum's system allows the payer, with the cooperation of the bank to verify which account the note was actually deposited to and the account will identify the real payee.

For Bitcoin, there is no such entity as a bank. Each node in the network can participate in accounting, generating a new block by a proof-of-work including new transactions broadcast. And the account ID of Bitcoin just corresponds to a Bitcoin address which is created privately instead of the user's identity. Though every transaction will be broadcast publicly, the Bitcoin network allows user to use different address for different payments which disable others to link two payments from different address together. With the multi-input transaction, some linking is unavoidable which necessarily reveals that their inputs were owned by the same owner.

Since the Bitcoin address can't identify the user, it is impossible to audit an individual's Bitcoin account which facilitates the black market, black payments for bribes, and blackmail. One of the well-known examples is that the worldwide WannaCry ransomware attack in May 2017 demanded ransom payments in Bitcoin [Wikb].

4.3.2 Trust assumptions

In Chaum's system, the security is based on that some particular problem and the inverse of some hash function are assumed hard. Meanwhile, we also trust the bank is honest.

For Bitcoin, all transactions and the ledger is public. The proof-of-work is also based on that the inverse of some particular hash function is assumed hard. By the longest chain principle and proof-of-work, the Bitcoin system ensures security as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes. The attacker who wants to change one block needs to redo the proof-of-work of the block and all the blocks after it and then catch up with the honest nodes. The nodes in the network are encouraged to be honest through incentives.

In the Bitcoin network, it is unnecessary to assume some central authority like a bank to be honest. Satoshi Nakamoto who propose Bitcoin says

> *The root problem with conventional currencies is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies is full of breaches of that trust.*

The philosophical idea behind Bitcoin is popular among libertarians and anarchists. They think the Bitcoin network is a good method to separate money from the government [Feu18].

4.3.3 Efficiency

In Chaum's system, the bank takes the responsibility to maintain the databases and serves as the blind signer. In terms of the user and the seller, Chaum's system is efficient.

For the Bitcoin network, every node can participate in generating a new block by finding proof-of-work. And there is only one node's work is accepted. One may assume that Chaum's system is more efficient totally than Bitcoin. Jones and Goodkind, and Berrens [JGB22] pointed out that Bitcoin mining contributes to energy-related climate damages. on average, each $1 in Bitcoin market value created was responsible for $0.35 in global climate damages. However, Khazzaka [Kha22] compares the Bitcoin system with classical electronic payment systems and states that Bitcoin consumes at least 28 times less energy and can run today with 60 times less energy than the classical system. This paper may prompt us to rethink the influence of Bitcoin on the environment.

References

[BD20] Elaine Barker and Quynh Dang. Nist special publication 800-57 part 1, revision 5. *NIST, Tech. Rep*, 16, 2020.

[BLL+22] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in) security of ros. *Journal of Cryptology*, 35(4):25, 2022.

[Bra94] Stefan Brands. Untraceable off-line cash in wallet with observers. In *Advances in CryptologyâCRYPTOâ93: 13th Annual International Cryptology Conference Santa Barbara, California, USA August 22–26, 1993 Proceedings 13*, pages 302–318. Springer, 1994.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto 82*, pages 199–203. Springer, 1983.

[CPS95] Jan L Camenisch, Jean-Marc Piveteau, and Markus A Stadler. Blind signatures based on the discrete logarithm problem. In *Advances in Cryptology-EUROCRYPT'94: Workshop on the Theory and Application of Cryptographic Techniques Perugia, Italy, May 9–12, 1994 Proceedings 13*, pages 428–432. Springer, 1995.

[DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *IEEE transactions on information theory*, 22(6), 1976.

[DKL15] Jannik Dreier, Ali Kassem, and Pascal Lafourcade. Formal analysis of e-cash protocols. In *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, volume 4, pages 65–75. IEEE, 2015.

[Feu18] Alan Feuer. The bitcoin ideology, 2018. https://web.archive.org/web/20180701222302/https://www.nytimes.com/2013/12/15/sunday-review/the-bitcoin-ideology.html.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II 38*, pages 33–62. Springer, 2018.

[FPS20] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 63–95. Springer, 2020.

[HLMN06] Charles B Haley, Robin C Laney, Jonathan D Moffett, and Bashar Nuseibeh. Using trust assumptions with security requirements. *Requirements Engineering*, 11:138–151, 2006.

[JGB22] Benjamin A Jones, Andrew L Goodkind, and Robert P Berrens. Economic estimation of bitcoin miningâs climate damages demonstrates closer resemblance to digital crude than digital gold. *Scientific Reports*, 12(1):14512, 2022.

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures. In *Annual International Cryptology Conference*, pages 150–164. Springer, 1997.

[Kha22] Michel Khazzaka. Bitcoin: Cryptopayments energy efficiency. *Available at SSRN*, 2022.

[KL20] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[Leg] Legrandin. Pycryptodome's documentation. https://pycryptodome.readthedocs.io/.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 2008.

[Plo21] Antoine Plouviez. *The security of the One-More Discrete-Logarithm assumption and Blind Schnorr Signatures*. PhD thesis, ENS Paris, 2021.

[PV05] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2005.

[Raz02] Rosehaslina Razali. The overview of e-cash: Implementation and security issues. *GSEC*, 2002.

[Sch01] Claus Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In *International Conference on Information and Communications Security*, pages 1–12. Springer, 2001.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in CryptologyâEUROCRYPTâ97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*, pages 256–266. Springer, 1997.

[Sta23] Secure Hash Standard. Federal information processing standard (fips) 186-4. *National Institute of Science and Technology*, 2023.

[Wag02] David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.

[Wika] Wikipedia. Computational complexity of mathematical operations. https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations.

[Wikb] Wikipedia. Wannacry ransomware attack. https://en.wikipedia.org/wiki/WannaCry_ransomware_attack.