Private coins extension with verifiable encryption

Oleg Fomenko
Distributed lab

August 2024

1 Introduction

The concept of storing values hidden within additive group elements is becoming increasingly popular. Nowadays, many different protocols are based on this idea. Most of them utilize the Pedersen commitment primitive, which allows for homomorphic commitment of users' balances and transfer amounts. However, one problem still remains – how can the receiver know the transfer balance if the sender has not informed them? This issue has the greatest impact on account-based blockchains, where it can be exploited to carry out spam attacks on users' accounts. For example, if protection mechanisms are not in place, a malicious user could submit multiple transfers without directly sharing the amounts with us. This would impact our ability to manage the balance in the future (as most of these protocols require the generation of range-proofs, which in turn necessitate knowing the committed amounts). One possible solution, explained in this paper, is to share the symmetrically encrypted amount along with the transfer amount commitment. This approach also requires a protocol to verify that the committed amount and the encrypted amount are the same. This is where verifiable encryption protocols come into play.

More precisely, given an amount $a \in \mathbb{F}$ committed in a Pedersen commitment with randomness r in the form $C = aH + rG \in \mathbb{G}$, we want to enable the sharing of a symmetric encryption of the value a using an arbitrary encryption scheme E, D. This should be done in such a way that there exist functions ϕ and $\hat{\phi}$ that satisfy the following properties:

$$e = \phi(E, s, a)$$
$$a = \hat{\phi}(D, s, e)$$

The protocol presented in this paper details how to generate such encryption (i.e., the functions ϕ and $\hat{\phi}$) and how to prove that the encrypted and committed amounts correspond to each other.

2 Preliminaries

We denote by $\mathbb{G}$ a cyclic group of prime order p written additively, which is, in practice, typically a subgroup of an elliptic curve. We write group elements in $\mathbb{G}$ with capital letters and scalars in $\mathbb{F} = \mathbb{F}_p$ with lower case letters. Vectors are written with bold letters. Let's also denote 1^λ a context-free protocol security parameter and $G(1^\lambda)$ a context-free security parameter's generator.

2.1 Discrete Logarithm Relation Problem

The discrete logarithm relation problem in $\mathbb{G}$ is *hard* if for a randomly chosen $a \in \mathbb{F}_p$, such that $A = aG$, given the point A, the probability of finding a from A is negligible.

2.2 Pedersen commitment

The **Pedersen commitment** [Ped91] schema is a cryptographic primitive that allows to commit to the scalar value in the additive group element. It is defined with respect to the two fixed additive group elements $G, H \in \mathbb{G}$ as follows:

$$Com(x, r) = xH + rG \in \mathbb{G}$$

where $r \in \mathbb{F}$ is a commitment blinding (big random scalar), $x \in \mathbb{F}$ – committed value, $G, H \in \mathbb{G}$ – group generators.

While defined over the additive group, Pedersen commitments allow addition between committed values that holds binding and hiding properties:

$$Com(x_1, r_1) + Com(x_2, r_2) = Com(x_1 + x_2, r_1 + r_2)$$

2.3 Σ-protocols

The Σ-protocol is a proof system to prove some relation $\mathbb{R}$ for the public parameter x and a private witness w using the following three algorithms with respect to the security parameter λ:

$\sigma^{\lambda}_{com}(x, w) \to \langle t, r \rangle$: generates a commitment t and some side information r for the tuple $\langle x, w \rangle$;

$\sigma^{\lambda}_{resp}(x, w, r, c) \to s$: generates a response for the received challenge c from verifier;

$\sigma^{\lambda}_{ver}(t, c, x, s) \to \{0, 1\}$: verifies that the response in a couple with the commitment satisfies the relation $\mathbb{R}$.

We also require that for the any two tuples $\langle t, c_1, s_1 \rangle$ and $\langle t, c_2, s_2 \rangle$, where $c_1 \neq c_2$, that satisfies the relation $\mathbb{R}$ such a polynomial-time procedure γ exists that accepts both of them and returns the value w. This requirement is crucial for constructing a verifiable encryption protocol based on the Σ-protocol.

2.4 Verifiable encryption

The verifiable encryption is an algorithm to prove that some secret encrypted value w satisfies the relation $\mathbb{R}$. More precisely, the relation $\mathbb{R}$ must be provable by some secure Σ-protocol. Then, the verifiable encryption protocol for the arbitrary cryptographic secure encryption E, D can be defined as follows:

Proving function: $P^{\lambda}(\sigma^{\lambda}_{com}, \sigma^{\lambda}_{res}, E^{\lambda}, x, w) \to \pi$

Verification function: $V^{\lambda}(\sigma^{\lambda}_{ver}, E^{\lambda}, x, \pi) \to \{0, 1\}$

Secret recover (decryption) function: $R^{\lambda}(\rho, x, \pi_0, \pi_1) \to w$

We imply that the proof π also contains the commitment and response for the Σ-protocol. Also, the recovery function R implies the usage of the Σ-protocol property of secret recovering with procedure ρ and the knowledge of two responses for different challenges.

The following protocol describes the simple version of verifiable encryption protocol from [CD00] (note, that any protocol with same requirements can be suitable):

Protocol 1 Simple verifiable encryption

Select keys $p_0,\ p_1 \leftarrow \mathbb{F}$

Put $t, r = \sigma^{\lambda}_{com}(x, w)$

Put $s_0, s_1 = \sigma^{\lambda}_{res}(x, w, r, 0),\ \sigma^{\lambda}_{res}(x, w, r, 1)$

Put $e_0, e_1 = E(p_0, s_0),\ E(p_1, s_1)$

Submit $\langle t, e_0, e_1 \rangle$ to the verifier and receive challenge $c \in \{0, 1\}$

Respond with $\langle p_c, s_c \rangle$

Verifier checks that $\sigma^{\lambda}_{ver}(t, c, s_c) = 1$ and $e_c = E(p_c, s_c)$.

To recover the encrypted value w the verifier can ask the prover to share the second private key $p_{\hat{c}}$ in a couple with $s_{\hat{c}}$. Then, by the properties of Σ-protocol the verifier will be able to restore w.

Using this protocol, prover can guess the correct answer with $\frac{1}{2}$ probability. By repeating the protocol $k = G(1^\lambda)$ times the probability falls down to $\frac{1}{2^k}$.

Note, that our paper uses the definition of encryption and decryption functions as $E, D : \mathbb{F}^2 \to \mathbb{F}$, but the only one requirement is that the messages field must be $\mathbb{F}$. The ciphertext and key fields can differ, but for simplicity, we also use $\mathbb{F}$.

3 Protocol

Let's assume we have an amount a stored in a Pedersen commitment of the form $C = aH + rG \in \mathbb{G}$, where $a \in \mathbb{F}$ is the amount and $r \in \mathbb{F}$ is the randomness. We may want to send, along with C, the value a encrypted with a symmetric key that can be decrypted by the recipient of the coins. This allows the recipient to recover the balance without a discrete log brute force. However, this approach requires an additional protocol to prove that the symmetrically encrypted value corresponds to the committed value.

More precisely, we want to prove the knowledge of such a, w that $C = aH + rG$ for public $G, H \in \mathbb{G}$ and $e = \phi(E, s, a)$ for an arbitrary cryptographic secure symmetric encryption E, D with symmetric key s. The symmetric key should be known only to the sender and receiver of the coins, so we utilize the Diffie-Hellman key exchange protocol to securely share it.

3.1 Σ-protocol

The corresponding Σ-protocol proves the knowledge of the openings to the commitment C by the prover:

Protocol 2 Σ-protocol

Input: a, r, C, G, H

The *Prover* selects random $x_1, x_2 \leftarrow \mathbb{F}_p$

Then, the *Prover* submits to the *Verifier*

$$X = x_1 H + x_2 G$$

The *Verifier* responds with the challenge ρ

The *Prover* calculates and submits responses to the *Verifier*:

$$\alpha_1 = x_1 - \rho a$$
$$\alpha_2 = x_2 - \rho r$$

The *Verifier* accepts if

$$X = \alpha_1 H + \alpha_2 G + \rho C$$

The **completeness** of the protocol follows from the definition. **Soundness**, and **zero-knowledge** of the described protocol follows from the corresponding proofs for the Schnorr identification protocol.

3.2 Verifiable encryption for presented Σ-protocol

The verifiable encryption (VE) protocol described earlier leverages on our Σ-protocol and extends it by executing $2k$ times, where $k = G(1^\lambda)$. For each VE protocol iteration we execute Σ-protocol twice that follows VE definition. The decryption of the amount is achieved through the properties of the Σ-protocol, where, by using responses to different challenges for the same commitment, we can reconstruct the private value.

Finally, the described solution is made non-interactive by using Fiat-Shamir heuristics. The resulting proof size is $9k \times \mathbb{F}$ values and $k \times \mathbb{G}$ values.

Protocol 3 VE proving

Select random $\mathbf{p} \leftarrow \mathbb{F}^{4\times k}$ and $\mathbf{x} \leftarrow \mathbb{F}^{2\times k}$

Using the receiver's symmetric key Diffie-Hellman share $s_r = g^u \in \mathbb{F}$ put $\mathbf{s} = (s_r)^{\mathbf{p}} \in \mathbb{F}^{4\times k}$

Put $X_i = x_{0,i}H + x_{1,i}G \in \mathbb{G}$

Put Σ-challenges $\rho_0 = Hash(G, H, C, \mathbf{X}, 0)$ and $\rho_1 = Hash(G, H, C, \mathbf{X}, 1)$

Calculate responses and their encryptions:

$$\begin{aligned}
\alpha_{0,i} &= x_{0,i} - \rho_0 a \\
\alpha_{1,i} &= x_{0,i} - \rho_1 a \\
\alpha_{2,i} &= x_{1,i} - \rho_0 r \\
\alpha_{3,i} &= x_{1,i} - \rho_1 r \\
e_{0,i} &= E(s_{0,i}, \alpha_{0,i}) \\
e_{1,i} &= E(s_{1,i}, \alpha_{1,i}) \\
e_{2,i} &= E(s_{2,i}, \alpha_{2,i}) \\
e_{3,i} &= E(s_{3,i}, \alpha_{3,i})
\end{aligned}$$

Put the verifiable encryption challenge $c = Hash(\rho_0, \rho_1, \mathbf{e})$

Add $\mathbf{e}$ to the proof π

For each bit b in first k bits of c add $\langle \alpha_{b,i},\ p_{b,i} \rangle$ and $\langle \alpha_{b+2,i},\ p_{b+2,i} \rangle$ to the proof π

Add $\mathbf{X}$ to the proof π

For each bit b in first k bits of c, put bit-inverse d and add $g^{p_{d,i}}$ to the proof π

Submit proof π to the *Verifier*

Protocol 4 VE verification

The *Verifier* performs the following operations to check the proof validness:

Restore Σ-challenges $\rho_0 = Hash(G, H, C, \mathbf{X}, 0)$ and $\rho_1 = Hash(G, H, C, \mathbf{X}, 1)$

Restore verifiable encryption challenge $c = Hash(\rho_0, \rho_1, \mathbf{e})$

For each bit b in first k bits of c check that:

$$\begin{aligned}
X_i = \alpha_{b,i}H + \alpha_{b+2,i}G + \rho_b C \wedge \\
e_{b,i} = E((s_r)^{p_{b,i}},\ \alpha_{b,i}) \wedge \\
e_{b+2,i} = E((s_r)^{p_{b+2,i}},\ \alpha_{b+2,i})
\end{aligned}$$

The **completeness** of the presented protocol is obvious. The **soundness** directly depends on the number of iterations $k = G(1^\lambda)$ (as described in VE definition paragraph) and the proofs can be found

in [CD00]. The **zero-knowledge** follows from the Σ-protocol, security of chosen encryption and hash function, and discrete logarithm hardness in $\mathbb{F}$: as long as an attacker cannot break the encryption or guess the Diffie-Hellman secret, the responses revealed from the Σ-protocol do not disclose any sensitive information to the attacker.

3.3 Amount recovery

To recover the encrypted value a the receiver can use the published $g^{p_{d,i}}$ in the proof π to restore the common Diffie-Hellman secret $s_{d,i} = (g^{p_{d,i}})^u$ and to decrypt the second response from each iteration $\alpha_{d,i}$. Then, using the following calculations receiver can evaluate the amount a:

$$a = \frac{\alpha_{0,i} - \alpha_{1,i}}{\rho_1 - \rho_0}$$

This equation is satisfied because:

$$\frac{\alpha_{0,i} - \alpha_{1,i}}{\rho_1 - \rho_0} = \frac{x_i - x_i + a(\rho_1 - \rho_0)}{\rho_1 - \rho_0} = \frac{a(\rho_1 - \rho_0)}{\rho_1 - \rho_0} = a$$

Even with the malicious sender, one of the recovered amounts will be correct with $1 - \frac{1}{2^k}$ probability.

The presented protocol also ensures that all symmetric keys were generated using Diffie-Hellman key exchange protocol by querying $p_{b,i}$ instead of $s_{b,i}$ each verification round. This allows prover to calculate the symmetric key by itself as $s_{b,i} = (s^r)^{s_{b,i}}$. Using the same deductions as for the amount recovery, the probability for malicious sender to generate non-Diffie-Hellman key while still passing the verification is $\frac{1}{2^k}$.

References

[Ped91] Torben Pryds Pedersen. *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*. 1991. URL: `https://link.springer.com/content/pdf/10.1007/3-540-46766-1_9.pdf`.

[CD00] Jan Camenisch and Ivan Damgard. *Verifiable Encryption, Group Encryption, and Their Applications to Separable Group Signatures and Signature Sharing Schemes*. 2000. URL: `https://link.springer.com/chapter/10.1007/3-540-44448-3_25`.