The Twisted ElGamal Encryption

Throughout this section, we work over an abstract cyclic group $(\mathbb{G}, +)$ of prime order $p \in \mathbb{N}$. In our actual implementation, we use the Ristretto group implementation over Curve25519 [2]. Following the implementation convention, we refer to elements in $\mathbb{Z}_p$ as *scalars* and elements in $\mathbb{G}$ as *points*.

1 Pedersen Commitments

Commitment definitions. A commitment scheme is a message encoding algorithm Commit that has the following syntax:

- $\mathsf{Commit}(x, r) \to C$: On input a scalar message x and a randomly generated "opening" r, the algorithm returns an encoding C that is generally referred to as a *commitment*.

A commitment scheme is secure if it satisfies the following two properties:

- *Binding*: Given a commitment C, it is hard to find two message-opening pairs (x_0, r_0), (x_1, r_1) such that
$$\mathsf{Commit}(x_0, r_0) = \mathsf{Commit}(x_1, r_1).$$
- *Hiding*: As long as an opening r is generated from a proper randomness source, a commitment C does not reveal any information about the committed message x.

The simplest example of a cryptographic commitment scheme is a scheme that uses a cryptographic hash function H:
$$\mathsf{Commit}(x, r) = H(x \| r),$$
where the message x and opening r are arbitrary bit-strings. A cryptographic hash function H is collision-resistant and therefore, it is hard to find two message-opening pairs (x_0, r_0) and (x_1, r_1) such that $H(x_0 \| r_0) = H(x_1 \| r_1)$. Furthermore, if the opening r is a sufficiently long, then the output of the hash does not reveal information about x.

Pedersen commitments. Pedersen commitments are defined with respect to two fixed group elements $G, H \in \mathbb{G}$. The commitment algorithm is defined as follows:

- $\mathsf{Commit}(x, r) \to C$: On input a scalar message $x \in \mathbb{Z}_p$ and a randomly generated opening $r \in \mathbb{Z}_p$, the algorithm returns
$$C = x \cdot G + r \cdot H \quad \in \mathbb{G}.$$

It is well-known that the Pedersen commitment scheme over any cryptographically robust group $\mathbb{G}$ satisfies both the security properties of binding and hiding. We refer to any introductory text on cryptography for the formal proofs of security.

2 Twisted ElGamal Encryption

The twisted ElGamal encryption scheme can be viewed as an augmentation of the standard Pedersen commitment scheme. As in the setting of Pedersen commitments, the twisted ElGamal encryption is defined with respect to two fixed group elements $G, H \in \mathbb{G}$ and are specified as follows:

- $\mathsf{KeyGen}() \to (\mathsf{pk}, \mathsf{sk})$: The key generation algorithm generates a random scalar $s \leftarrow_{\mathsf{R}} \mathbb{Z}_p$. It computes $P = s^{-1} \cdot H$ and sets
$$\mathsf{pk} = P, \quad \mathsf{sk} = s.$$
- $\mathsf{Encrypt}(\mathsf{pk}, x) \to \mathsf{ct}$: The encryption algorithm takes in a public key $\mathsf{pk} = P \in \mathbb{G}$ and a message $x \in \mathbb{Z}_p$ to be encrypted. It samples a random scalar $r \leftarrow_{\mathsf{R}} \mathbb{Z}_p$ and then computes the following computes:
  1. *Pedersen commitment*: $C = r \cdot H + x \cdot G$,
  2. *Decryption handle*: $D = r \cdot P$.

  It returns $\mathsf{ct} = (C, D)$.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to x$: The decryption algorithm takes in a secret key $\mathsf{sk} = s$ and a ciphertext $\mathsf{ct} = (C, D)$ as input. It computes
$$C = C - s \cdot D \quad \in \mathbb{G},$$
and then solves the discrete log problem to recover $x \in \mathbb{Z}_p$ for which $x \cdot G = P$.

Correctness. The correctness of the decryption algorithm can be easily verified. Let $\mathsf{pk} = P$, $\mathsf{sk} = s$ be a public-secret key pair and let $\mathsf{ct} = (C = r \cdot H + x \cdot G, D = r \cdot P)$ be a properly encrypted ciphertext of a scalar message $x \in \mathbb{Z}_p$. Then, a proper decryption of the ciphertext produces:

$$\begin{aligned} C - s \cdot D &= (rH + xG) - s \cdot (rP) \\ &= rH + xG - r \cdot (sP) \\ &= rH + xG - rH \\ &= xG \end{aligned}$$

Assuming that the discrete log is solved correctly, the decryption algorithm recovers the original message $x \in \mathbb{Z}_p$.

Security. The security of the twisted ElGamal encryption follows from the standard Decision Diffie-Hellman (DDH) assumption. We refer to [1] for the formal proof of security.

3 Computing Discrete Log.

Let $\mathbb{G}$ be a cyclic group of prime order $p \in \mathbb{N}$, and let $G \in \mathbb{G}$ be any fixed point in $\mathbb{G}$ generally referred to as the "generator" for $\mathbb{G}$. Then, any point $C \in \mathbb{G}$ can be represented as a multiple of G:

$$x \cdot G = C.$$

The discrete log problem is the computational task of recovering x given a generator G and a target C:

- **Input**: generator $G \in \mathbb{G}$ and target $C \in \mathbb{G}$
- **Output**: $x \in \mathbb{Z}_p$ such that $x \cdot G = C$

If C is an arbitrary random element in the group G, then this problem is well-known to be hard to compute. However, if there is a guarantee that the unique scalar $x \in \mathbb{Z}_p$ for which $C = x \cdot G$ is a small number (e.g. $0 \leq x < 2^{32}$), then the problem can be solved efficiently.

Suppose that x is a 32-bit number. The simplest way to solve the discrete log problem is to enumerate through all possible values $0 \leq x < 2^{32}$ and test for the relation $C = x \cdot G$. Although this brute-force way of computing discrete log is in the realm of feasibility with modern hardware, we can solve the problem much more efficiently by relying on the space-time tradeoff. Let $x_{\mathsf{lo}}, x_{\mathsf{hi}}$ be two numbers represented by the least and most significant 16-bits of x such that $x = x_{\mathsf{lo}} + 2^{16} \cdot x_{\mathsf{hi}}$. Then the discrete log relation can be represented as

$$\begin{aligned} C &= x \cdot G \\ &= (x_{\mathsf{lo}} + 2^{16} \cdot x_{\mathsf{hi}}) \cdot G \\ &= \underbrace{x_{\mathsf{lo}} \cdot G}_{\text{online}} + \underbrace{2^{16} \cdot x_{\mathsf{hi}} \cdot G}_{\text{offline}} \end{aligned}$$

Using this relation, one can solve the discrete log problem by first computing all possible values of $2^{16} \cdot x_{\mathsf{hi}} \cdot G$ for $x_{\mathsf{hi}} = 0, 1, \ldots, 2^{16} - 1$ and then solving for x_{lo}:

$\mathsf{ComputeDiscreteLog}(G, C)$:

1. Instantiate a look-up table (e.g. HashMap). For $x_{\mathsf{hi}} = 0, \ldots 2^{16} - 1$, store the following key-value pairs
   - `key`: $2^{16} \cdot x_{\mathsf{hi}} \cdot G$
   - `value`: x_{hi}
2. For $x_{\mathsf{lo}} = 0, \ldots, 2^{16} - 1$, check whether a key $C - x_{\mathsf{lo}} \cdot G$ exists in the look-up table. If it exists, then take the table entry x_{hi} and return $x = x_{\mathsf{lo}} + 2^{16} \cdot x_{\mathsf{hi}}$.

We note that the look-up table in step 1 above is independent of the discrete log target $C \in \mathbb{G}$. Therefore, the look-up table can be pre-computed once offline and can be re-used for multiple instances of discrete log with a fixed generator $G \in \mathbb{G}$. The look-up table stores 2^{16} entries of 16-bit numbers, which translates to around 130 kilobytes. With this amount of pre-computation, any discrete log challenge can be solved in a matter of seconds.

We note that the algorithm above divides the 32-bit number x specifically into two 16-bit numbers x_{lo} and x_{hi}. This choice of numbers is purely for simplicity; the numbers x_{lo} and x_{hi} can be chosen to be arbitrary ℓ_{lo} and ℓ_{hi}-bit numbers for which $\ell_{\mathsf{lo}} + \ell_{\mathsf{hi}} = 32$. The numbers ℓ_{lo} and ℓ_{hi} define a space-time trade-off in computing the discrete log problem. For instance, by setting ℓ_{lo} smaller and ℓ_{hi} larger, the discrete log problem can be solved in less than a second with larger pre-computed data.

References

[1] Chen, Y., Ma, X., Tang, C., and Au, M. H. Pgc: Decentralized confidential payment system with auditability. In *European Symposium on Research in Computer Security* (2020), Springer, pp. 591–610.

[2] Lovecruft, I. A., and de Valence, H. `curve25519-dalek`. https://github.com/dalek-cryptography/curve25519-dalek, 2021.