Verifiable Encryption using Halo 2

QEDIT

Abstract. This work aims to construct a cryptographic system that allows a prover to prove some property of a message m ($(m, \delta) \in \mathcal{R}$ for a given statement δ), while the message is given in an encrypted form. Verifiable encryption is a useful tool to achieve this goal. Our contribution lies in the introduction of a generic construction for verifiable encryption protocols from two main components: a proof system and a public key encryption scheme. Furthermore, we prove that the newly constructed protocol satisfies the security properties for verifiable encryption if the underlying building blocks are secure. For the practical realization of this system, we leverage the Halo 2 proof system to create and verify proofs for our constraint systems. For the implementation of the public key encryption, we have selected for the ElGamal Encryption scheme due to its compatibility and effectiveness within the established framework.

1 Task Overview

We list two real life problems that we aim to solve in the work.

Task 1 is a warm up task for the prover to prove to the verifier that she knows a message m that encrypts to a ciphertext C.

Task 2 is for the prover to prove to the verifier that she knows a message m that encrypts to a ciphertext C. Additionally, the message is a private key of a digital signature scheme.

2 Preliminary

We denote λ as the security parameter and $\mathsf{negl}(\lambda)$ as the negligible function in λ. The operation of scalar multiplication of a group element P by a scalar a is denoted as $[a]P$. Furthermore, the notation $a \xleftarrow{\$} S$ represents randomly selecting a value a from the set S.

2.1 ECC Preliminary

Elliptic Curve Cryptography (ECC) is a Public Key Cryptography (PKC) technique based on the algebraic structure of elliptic curves over finite fields. It provides a compact and efficient cryptographic system compared to traditional PKC methods like RSA. Notably, ECC offers equivalent security to RSA/Paillier with significantly shorter key lengths. For example, a 256-bit ECC key offers security equivalent to a 3072-bit RSA/Paillier key. In this work, we employ ECC to develop PKC protocols.

Selecting Elliptic Curves for Public Key Encryption Elliptic Curve Cryptography (ECC) [Nak21, BCLN16] can utilize various underlying elliptic curves. An affine curve is defined by the equation $y^2 = x^3 + ax + b$. Different curves offer varying levels of security (cryptographic strength), performance (speed), and key length, and may also involve distinct algorithms. We should consider these factors when selecting a curve for public key encryption. The security of elliptic curve cryptography relies on the difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP).

Selecting Elliptic Curves for Halo 2 Proof System Pasta curves [Com21], which are defined by the equation $y^2 = x^3 + 5$, have been specifically chosen for integration into the Halo 2 framework. This choice significantly enhances the efficiency and security of recursive proof systems, such as PLONK, on which Halo 2 is based. To effectively employ the Halo 2 proof system for our projects, it is essential to adjust encryption operations to be compatible with pasta curves. These curves are established over 255-bit prime fields, offering a 126-bit level of security against Pollard's rho attacks. This level of security makes them highly suitable for public key encryption systems, such as ElGamal encryption.

2.2 ECElGamal

Let E be an elliptic curve over a finite field $\mathbb{F}_p$, G on E to be the generator of the elliptic curve group of order q. We set the public parameter $\mathsf{pp} = (E, \mathbb{F}_p, G, p, q)$.

Definition 1. *The Elliptic Curve ElGamal encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a public key encryption scheme that:*

- Gen *is the key-generation algorithm that takes the public parameter* pp *as input and outputs a pair of keys* (pk, sk) *for public key* pk *and private key* sk*.*
  1. *Select a random private key* $sk \xleftarrow{\$} \mathbb{F}_q$*.*
  2. *Compute the public key* $pk = [sk]G$*.*
  3. *Return the public key* pk *and the private key* sk*.*
- Enc *is the encryption algorithm that takes the public key* pk *and a plaintext message* $M \in E(\mathbb{F}_p)$ *and outputs a ciphertext* C*.*
  1. *Choose a random element* $r_{\mathsf{enc}} \xleftarrow{\$} \mathbb{F}_q$*.*
  2. *Compute the ephemeral element* $C_1 = [r_{\mathsf{enc}}]G$*.*
  3. *Compute the ciphertext* $C_2 = M + [r_{\mathsf{enc}}]pk$*.*
  4. *Return the ciphertext* $C = (C_1, C_2)$*.*
- Dec *is the decryption algorithm that takes the private key* sk *and a ciphertext* C *and outputs a plaintext message* m *or an error symbol* $\perp$*.*
  1. *Pharse* $C = (C_1, C_2)$*.*
  2. *Compute* $M = C_2 - [sk]C_1$*.*
  3. *Return* M*.*

A notation. In the upcoming section, while conducting the proof in verifiable encryption, the encryptor will need to utilize the randomness r_{enc} used in the encryption algorithm. We denote the pair (C, r_{enc}) by $\mathsf{Enc}'_{pk}(M; r_{\mathsf{enc}})$, indicating the encryption of message M with public key pk and randomness r_{enc}.

2.3 Encode a Message into a Curve Point

In real-world applications, messages can take various forms, such as strings, finite field elements, etc. To facilitate secure data transmission using ElGamal encryption, we introduce encoding and decoding algorithms. These algorithms create an invertible mapping between the actual message space and the ElGamal message space, denoted as $E(\mathbb{F}_p)$. This mapping allows for the encryption of messages in a form suitable for secure transmission.

Suppose the message space is $\mathbb{F}_p$, the curve E is defined by the equation $y^2 = x^3 + ax + b$. we outline the construction of encode and decode functions between $\mathbb{F}_p$ and $E(\mathbb{F}_p)$ as follows:

Definition 2. Encode *and* Decode *algorithms are used to transform data between* $\mathbb{F}_p$ *and* E*. The following arithmetic operations are performed within the finite field* $\mathbb{F}_p$*, we omit writing* $\bmod p$ *in each calculation.*

- Encode *takes a message* $m \in \mathbb{F}_p$ *and outputs a point* $P_m \in E(\mathbb{F}_p)$*. The encode overview is:*

$$m \overset{Step\ 1-2}{\rightarrow} x_m \overset{Step\ 3-4}{\rightarrow} (x_m, y) \overset{Step\ 5}{\rightarrow} P_m$$

  1. *Select a random element* $r_{\mathsf{encode}} \xleftarrow{\$} \mathbb{F}_p$*.*
  2. *Compute* $x_m = m + r_{\mathsf{encode}}$*.*
  3. *Calculate* $y_{square} = x_m^3 + ax + b$*.*
  4. *Calculate the square root* $y = \sqrt{y_{square}}$*; if a square root does not exist, revert to step 1.*
  5. *Convert the affine coordinate* (x_m, y) *to a curve point* P_m *in the form of Jacobian coordinate.*
  6. *Return* $(P_m, r_{\mathsf{encode}})$*.*
- Decode *takes a point* P'_m *on the elliptic curve* E *and a randomness* r_{encode}*, and outputs a message* $m' \in \mathbb{F}_p$*. The decode overview is:*

$$P'_m \overset{Step\ 1}{\rightarrow} (x_m, y) \overset{Step\ 2}{\rightarrow} m'$$

  1. *Convert* P'_m *to its affine coordinate, denoted as* (x_m, y)*, where* x_m *is the x-coordinate of the affine point encoding the message.*
  2. *Compute* $m' = x_m - r_{\mathsf{encode}}$
  3. *Return* m'

Remark. Any other data type can be directly converted to $\mathbb{F}_p$ and then encoded to a point on the curve using the aforementioned algorithms. To ensure an embedding from these data types to $\mathbb{F}_p$ is feasible, the data type space cannot be bigger than that of $\mathbb{F}_p$. The requirement is that the data types should not exceed 255 bits, given that p is a 255-bit prime number.

2.4 Real Application Process

To encrypt a generic message m, assuming $m \in \mathbb{F}_p$. The process of encode, encryption, decryption and decode, as outlined in Fig. 1, address this task.

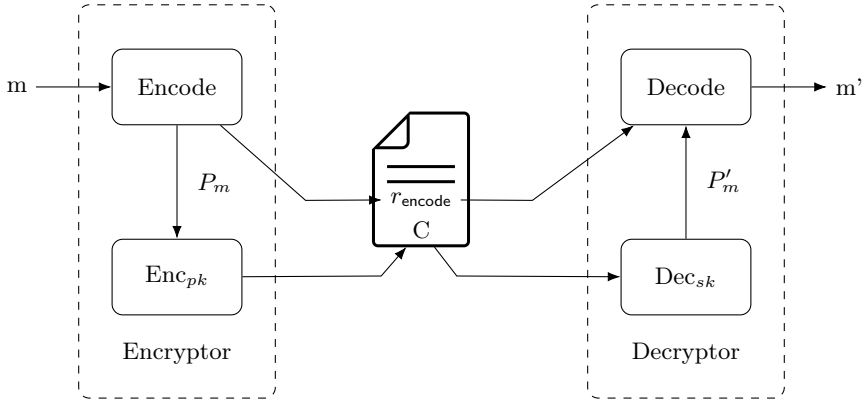

Fig. 1. The encode - encryption - decryption - decode process.

To evaluate the security of the system mentioned above, we define an extended version of the ElGamal encryption, as detailed in Definition 3.

Definition 3. *An extended ElGamal encryption* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is constructed from the original ElGamal encryption (see Definition 1) and* $(\mathsf{Encode}, \mathsf{Decode})$ *(see Definition 2):*

- Gen *is the same as in the original ElGamal encryption.*
- Enc *takes the public key pk and a plaintext message* $m \in \mathbb{F}_p$*, and outputs a ciphertext* C *and a randomness* r_{encode}*.*
  1. *Compute* $\mathsf{Encode}(m)$ *and obtain* $(P_m, r_{\mathsf{encode}})$*.*
  2. *Compute* $C = \mathsf{ElGamal.Enc}_{pk}(P_m)$*.*
  3. *Return* (C, r_{encode})*.*
- Dec *is the decryption algorithm that takes the private key sk, a ciphertext* C *and a randomness* r_{encode} *and outputs a plaintext message* m *or an error symbol* $\perp$*.*
  1. *Compute* $P'_m = \mathsf{ElGamal.Dec}_{sk}(C)$*.*
  2. *Compute and return* $m' = \mathsf{Decode}(P'_m, r_{\mathsf{encode}})$*.*

Next, we show that data known publicly does not leak any information about the underlying message.

Theorem 4. *If DDH is hard relative to* $E(\mathbb{F}_p)$*, then the extended ElGamal encryption* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is CPA-secure.*

The proof of Theorem 4 is provided in Appendix B.

2.5 Halo 2 Proof System

Halo 2 [BGH19, Com21] is designed to work with elliptic curves E over finite field $\mathbb{F}_p$. The group order of $E(\mathbb{F}_p)$ should be a prime for security reasons. The selection of Pasta curves is critical to the efficiency of the recursive proof system [Com21], which is essential for its scalability and efficiency.

The security properties of Halo 2 are outlined in the Halo 2 book [hal24], where it is described as offering perfect completeness, witness extended emulation, and perfect special honest-verifier zero knowledge. Witness extended emulation represents an enhanced form of knowledge soundness.

3 Verifiable Encryption

Verifiable encryption is a protocol that allows an encryptor/prover to provide a proof that a plaintext m has been encrypted under a specific public key pk. This proof allows a verifier to check that a ciphertext represents a valid encryption of a particular message under a given public key, all while maintaining the confidentiality of the message itself. Additionally, when applied to a relation $\mathcal{R}$, verifiable encryption enables the encryptor/prover to prove some property of the plaintext m ($(m, \delta) \in \mathcal{R}$ for a given statement δ).

Within such a protocol, both the prover and the verifier share common inputs: the encryption scheme's public key pk, a statement δ related to the message m, a ciphertext C corresponding to the message m and defined constraint systems. The encryptor/prover additionally has private inputs including the message m and the randomness r_{enc} used by the encryption. The prover's task is to create a proof that demonstrates knowledge of a witness, which, when encrypted, results in the ciphertext C. The verifier will assess the proof with respect to the public inputs, and ultimately decide to accept or reject the proof.

3.1 Constructing a Verifiable Encryption Protocol

We propose a generic construction for a protocol that integrates a proof system with a public key encryption scheme, as detailed in Definition 5. Subsequently, we validate the security properties of the proposed protocol, confirming it is a verifiable encryption [CS03, Definition 1].

Definition 5. *We construct a protocol from a proof system $(\mathcal{P}, \mathcal{V})$, a public encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, a relation generator $\mathcal{G}'$, and a reconstruction algorithm recon.*

Setup We run the setup to generate the witness and public inputs to the proof system as follows:

- *Let (pk, sk) be a public and private key pair of the encryption scheme,*
- *$(m, \delta) \in \mathcal{R}$ is the relation with respect to which we want to verifiably encrypt.*
- *Compute a ciphertext $(C, r_{\mathsf{enc}}) \leftarrow \mathsf{Enc}'_{pk}(m)$.*
- *Set the witness be $w = (m, r_{\mathsf{enc}})$, the public input be $x = (C, pk, \delta)$, and define the relation $\mathcal{R}_{\mathsf{proof\_system}} = \{(w, x) : C = \mathsf{Enc}_{pk}(m; r_{\mathsf{enc}}) \wedge (m, \delta) \in \mathcal{R}\}$.*

– *The prover takes the witness and public inputs pair (w, x) as input and generates a proof z for the constraint systems $C = \mathsf{Enc}_{pk}(m; r_{\mathsf{enc}})$ and $(m, \delta) \in \mathcal{R}$.*
– *The verifier given the public input x, proof z and the relation $\mathcal{R}_{\mathsf{proof\_system}}$, outputs accept or reject.*

Compatibility. When integrating a public key encryption with a proof system, the choice of prime field and elliptic curve must be compatible or easily translatable. This means that the witness and public input, as outlined for the proof system, should be readily compatible or easily translatable to elements that align with the prime field or elliptic curve employed by the proof system. Essentially, this requirement ensures that the cryptographic components operate seamlessly within the same mathematical framework, allowing for efficient and secure implementation of the combined systems.

Definition 6 (Simulatable relation). *We say a relation is simulatable if for all $(m, \delta) \in \mathcal{R}$ it holds that $\delta = f(m)$, and there exists a simulator Sim that the distributions it generates are indistinguishable from those of the actual relation generation*

– $\{\delta : m \leftarrow \mathcal{M}, \delta = f(m)\}$
– $\{\delta : \delta \leftarrow \mathsf{Sim}(\lambda)\}$

Simulatable relation is crucial for demonstrating the security of cryptographic protocols, especially in the context of zero-knowledge proofs, where it is essential to show that no additional information about the message (m in this case) is leaked from what is known (the statement δ).

Let's look at a concrete example where the relation is based on the Dlog problem, and the message is chosen at random from $\mathbb{F}_p$. In this scenario, there exists a simulator Sim that is randomly generating an element from $E(\mathbb{F}_p)$. The key point of analysis here is that the distributions produced by the real relation (where a genuine message m is the witness of the Dlog relation) and those generated by Sim (simulating the process without actual knowledge of the message) are indistinguishable:

– $\{\delta : m \leftarrow \mathbb{F}_p, \delta = [m]G\}$
– $\{\delta : \delta \leftarrow E(\mathbb{F}_p)\}$

Next, we show a generic result that prove the protocol constructed in Definition 5 is a verifiable encryption if the underlying proof system, PKE and relation satisfy their respective security requirements.

Theorem 7. *If a proof system $(\mathcal{P}, \mathcal{V})$ satisfies completeness, soundness, and special honest-verifier zero knowledge, a public key encryption $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is CPA-secure, and a relation $\mathcal{R}$ is simulatable then the protocol as constructed in Definition 5 using this proof system together with this public key encryption for the relation $\mathcal{R}$ is a verifiable encryption protocol [CS03, Definition 1].*

The proof of Theorem 7 is provided in Appendix C.

Real Construction for Implementation. Let's consider a specific instance of the protocol as constructed in Definition 5, we utilize a combination of the following components, each with its distinct security properties:

– Halo 2 proof system, The security properties of this system are detailed in Section 2.5.
– extended ElGamal encryption, CPA security for this encryption is given in Theorem 4.
– DLog relation $\mathcal{R} = \{(m, \delta) : m \leftarrow \mathbb{F}_p, \delta = [m]G\}$ is simulatable.

According to Theorem 7, these components collectively form a verifiable encryption protocol. Beyond security considerations, the extended ElGamal encryption and the Discrete Logarithm (Dlog) relation are compatible with the Halo 2 proof system. In the subsequent sections, we will leverage this protocol to address our specific tasks, demonstrating its utility in practical cryptographic applications.

3.2 Verifiable Encryption without Relation $\mathcal{R}$

To prove a ciphertext C is an encryption of a message $m \in \mathbb{F}_p$ under a public key pk. We utilize the aforementioned approach, where the extended ECElGamal encryption scheme (described in Section 2.4) can be used to encrypt m.

Halo 2 proof system is provided with the following information to validate the encryption.

– The private witness input would be $w = (m, P_m, r_{\mathsf{enc}})$
– The public input would be $x = (C = (C_1, C_2), r_{\mathsf{encode}}, pk, G)$.
– The NP relation $\mathcal{R}_{\mathsf{proof\_system}} = \{(w, x) : P_m = \mathsf{Encode}(m; r_{\mathsf{encode}}) \wedge C = \mathsf{ElGamal.Enc}_{pk}(P_m; r_{\mathsf{enc}})\}$

More precisely, the constraint systems can be split into the following equations:

1. $\mathsf{Encode}(m; r_{\mathsf{encode}}) = P_m$, refer to the encode algorithm in Section 2.3, this relation is to verify that P_m correctly represents the message m through:
   (a) $m + r_{\mathsf{encode}} = P_m.x$
   (b) $P_m.x^3 + 5 = P_m.y^2$.
2. $C = \mathsf{ElGamal.Enc}_{pk}(P_m)$, refer to the encryption algorithm in Section 2.2, this relation is to verify that C is the result of encrypting P_m under pk:
   (a) $C_1 = [r_{\mathsf{enc}}]G$
   (b) $C_2 = P_m + [r_{\mathsf{enc}}]pk$

Remark. The above system addresses the verification of a ciphertext as a proper encryption under a specific public key. However, it does not extend to validating the intrinsic properties of the message m itself. PKE allows the encryption of any arbitrary message $m*$ and demonstrate knowledge of this message. The assurance that the underlying message meets specific criteria remains outside the scope of the above approach.

Verify multiple message blocks. Suppose a prover has a long message, it divides the message into n blocks, each block denoted as $block_i$ for $i \in \{0, 1, \ldots, n-1\}$. The prover encrypts each block and obtain n ciphertexts $C_1, \ldots, C_n$. As a result, the prover has n NP statements and seeks to convince the verifier that all of these statements are true. A trivial solution for the prover is to prove $((block_i, r_{\mathsf{enc},i}), C_i) \in \mathcal{R}_{\mathsf{proof\_system}}$ for all $i \in [n]$.

3.3 Verifiable Encryption with Relation $\mathcal{R}$

In this section, we focus on a specific relation $\mathcal{R} = \{(m, pk_{\mathsf{DSA}}) : pk_{\mathsf{DSA}} = [m]G\}$. The relation requires the message is a private key of a DSA scheme with respect to the public key pk_{DSA}.

Apart from proving the ciphertext is a valid encryption of m, we additionally prove that $(m, pk_{\mathsf{DSA}}) \in \mathcal{R}$. The Halo 2 proof system needs to be equipped with additional inputs, specifically pk_{DSA} and the constraint system $pk_{\mathsf{DSA}} = [m]G$.

- The private witness input would be $w = (m, P_m, r_{\mathsf{enc}})$
- The public input would be $x = (C = (C_1, C_2), r_{\mathsf{encode}}, pk_{\mathsf{enc}}, pk_{\mathsf{DSA}}, G)$.
- The NP relation $\mathcal{R}_{\mathsf{proof\_system}} = \{(w, x) : P_m = \mathsf{Encode}(m; r_{\mathsf{encode}}) \wedge C = \mathsf{ElGamal.Enc}_{pk_{\mathsf{enc}}}(P_m; r_{\mathsf{enc}}) \wedge pk_{\mathsf{DSA}} = [m]G\}$

More precisely, the constraint systems can be split into the following equations:

1. $\mathsf{Encode}(m; r_{\mathsf{encode}}) = P_m$, refer to the encode algorithm in Section 2.3, this relation is to verify that P_m correctly represents the message m through:
   (a) $m + r_{\mathsf{encode}} = P_m.x$
   (b) $P_m.x^3 + 5 = P_m.y^2$.
2. $C = \mathsf{ElGamal.Enc}_{pk_{\mathsf{enc}}}(P_m)$, refer to the encryption algorithm in Section 2.2, this relation is to verify that C is the result of encrypting P_m under pk_{enc}:
   (a) $C_1 = [r_{\mathsf{enc}}]G$
   (b) $C_2 = P_m + [r_{\mathsf{enc}}]pk_{\mathsf{enc}}$
3. $pk_{\mathsf{DSA}} = [m]G$

3.4 Implementation

The implementation details and source code are referenced in the GitHub repository [Qi24]. Table 1 outlines the runtime performance and proof size metrics for our implementation, providing a quantitative analysis of the efficiency and scalability of our cryptographic solution.

Process	Runtime (s)	Proof length (byte)	Function
parameter generation	6.58		Params::new
vk generation	1.32		plonk::keygen_vk
pk generation	0.74		plonk::keygen_pk
proof generation	4.89		plonk::create_proof
verify	0.10		plonk::verify_proof
proof length		4000	

Table 1. The runtime and size are tested for Verifiable Encryption with Relation $\mathcal{R}$.

4 Additional Explorations: Approaches That Did Not Work

Well-established public key encryption schemes include RSA, Paillier and ElGamal encryption schemes. When integrating a public key encryption with the Halo 2 proof system, compatibility, security, and efficiency are essential to consider. For compatibility, the witness and public inputs generated from the encryption must be aligned with the prime field or elliptic curve selected by Halo 2. This requirement significantly narrows the choice of encryption schemes to those defined over elliptic curves due to the mathematical structures Halo 2 is built upon.

An initial exploration involved evaluating the EC Paillier encryption scheme [Gal02], to assess its suitability for use in verifiable encryption. However, the elliptic curve used in the Paillier encryption is constructed over a ring R and the group order $E(R)$ is not prime. This characteristic renders the EC Paillier encryption scheme incompatible with the requirements of the Halo 2 system, as elucidated in Section 2.5.

Subsequently, the attempt shifted towards the EC ElGamal encryption scheme. The analysis presented in Section 2 concludes that EC ElGamal is compatible with the Halo 2 proof system

References

BCLN16. J. W. Bos, C. Costello, P. Longa, and M. Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *J. Cryptogr. Eng.*, 6(4):259–286, 2016.

BGH19. S. Bowe, J. Grigg, and D. Hopwood. Halo: Recursive proof composition without a trusted setup. *IACR Cryptol. ePrint Arch.*, page 1021, 2019.

Com21. E. C. Company. The pasta curves for halo 2 and beyond, 2021. `https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/`.

CS03. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, *Lecture Notes in Computer Science* 2729, pages 126–144. Springer, 2003.

Gal02. S. D. Galbraith. Elliptic curve paillier schemes. *J. Cryptol.*, 15(2):129–138, 2002.

hal24. The halo2 book, 2024. `https://zcash.github.io/halo2/design/protocol.html`.

Nak21. S. Nakov. Elliptic curve cryptography (ecc), 2021. `https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc`.

Qi24. QED-it. Github halo2 verifiable encryption repository, 2024. `https://github.com/QED-it/halo2_verifiable_encryption/tree/main`.

A Implementation Details

In elliptic curve cryptography, the representation of points on an elliptic curve can significantly impact the efficiency of cryptographic operations. Two common coordinate systems used are affine and Jacobian coordinates.

In affine coordinates, a point P on an elliptic curve is represented by two field elements (x, y) that satisfy the elliptic curve equation of the form: $y^2 = x^3 + ax + b$.

To enhance computational efficiency, particularly in point multiplication, Jacobian coordinates are employed. In this system, a point P is represented by three field elements (X, Y, Z). The conversion from Jacobian to affine coordinates is given by: $x = X/Z^2, y = Y/Z^3$. Using Jacobian coordinates allows for certain operations to be performed without field inversions, which are costly in terms of computational resources. This feature significantly accelerates the process of point multiplication, a crucial operation in many cryptographic protocols.

Two points in Jacobian coordinates, (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2), are considered to represent the same point in affine coordinates on the elliptic curve if:

$$x = X_1/Z_1^2 = X_2/Z_2^2, y = Y_1/Z_1^3 = Y_2/Z_2^3$$

In our source code [Qi24]. The specific domains of pasta curves, including their parameters and the elliptic curve equations they satisfy, are detailed in Table 2. Additionally, Table 3 lists frequently used functions in the context of these curves, providing a comprehensive reference for their application in cryptographic operations. This structured documentation is essential for understanding the implementation of cryptographic protocols that utilize pasta curves.

Domain	pallas	vesta
Fp	Base	Scalar
Fq	Scalar	Base
Ep	Point	
Eq		Point
EpAffine	Affine	
EqAffine		Affine

Table 2. Domains of Pasta curves

B Proof of Theorem 4

Proof. We prove the real or random variant of CPA security for the extended ElGamal encryption.

function	input	output	Note
pallas::Base::from(input)	u64/bool	Fp	
pallas::Base::from_raw(input)	[u64; 4]	Fp	
input.to_repr()	Fp/Fq	[u8; 32]	prime field element → bytes
pallas::Base::from_repr(input)	[u8; 32]	Fp	bytes → prime field element
pallas::Scalar::from_repr(input)	[u8; 32]	Fq	bytes → prime field element
input.unwrap()		underlying value	but panics if it is not Some
pallas::Affine::from_xy(input)	x: pallas::Base, y: pallas::Base	EpAffine	$x, y \to (x, y)$
input.to_curve()	EpAffine	Ep	$(x, y) \to (x, y, 1)$
input.to_affine()	Ep	EpAffine	$(x, y, z) \to (x', y')$
.to_affine().coordinates().unwrap().x()	Ep	Fp	

Table 3. Data type transformation functions

In the real or random confidentiality game, the adversary is challenged to differentiate between a ciphertext that encrypts a message under the public key encryption scheme and a ciphertext that is generated randomly. The security of the encryption scheme is established if the adversary cannot make this distinction better than random guessing.

We construct a reduction that simulates the responses of queries made by the adversary. The reduction takes a DDH tuple $([a]G, [b]G, [c]G)$ as input and embeds the DDH tuple into the extended ElGamal scheme's public key and ciphertext components as follows:

- The public key pk is set to $[a]G$
- The first part of the ciphertext C_1 is $[b]G$
- The second part of the ciphertext C_2 is formulated as $p_m + [c]G$, where p_m is the encoded message point.

When $([a]G, [b]G, [c]G)$ constitutes a real DDH tuple ($c = ab$), the reduction perfectly simulates the real game for the adversary.

Conversely, when the DDH tuple $([a]G, [b]G, [c]G)$ is random(c is an independent random element), the reduction simulates $(pk, C, r_{\mathsf{encode}})$ as $([a]G, ([b]G, p_m + [c]G), r_{\mathsf{encode}})$ where a, b, c and r_{encode} are independent random values. This means the reduction perfectly simulates the random game for the adversary.

Now, the adversary's challenge is to distinguish between these two simulated games. If the adversary cannot distinguish between a real DDH tuple and a random one, it implies they cannot distinguish between a real encryption and a random ciphertext in the extended ElGamal scheme, underpinning the scheme's CPA security.

This reduction demonstrates that breaking the CPA security of the extended ElGamal encryption would imply solving the DDH problem, which is assumed to be hard. Therefore, as long as the DDH assumption holds, the extended ElGamal encryption scheme can be considered secure against CPA attacks.

C Proof of Theorem 7

For comprehensive definitions of correctness, soundness, and special honest-verifier zero-knowledge in the context of verifiable encryption, please refer to [CS03, Definition 1]. Let's assume the proof system $(\mathcal{P}, \mathcal{V})$ satisfies completeness, soundness, and special honest-verifier zero knowledge. We prove the security properties of the protocol as constructed in Definition 5.

Correctness. For all $(pk, sk) \in \mathsf{Gen}$, for all $(m, \delta) \in \mathcal{R}$, for all $(C, r_{\mathsf{enc}}) \in \mathsf{Enc}'_{pk}(m)$. We consider:

- The witness $w = (m, r_{\mathsf{Enc}})$
- The statement $x = (C, pk, \delta)$, and
- The relation to the proof system $\mathcal{R}_{\mathsf{proof\_system}} = \{(w, x) : C = \mathsf{Enc}_{pk}(m; r_{\mathsf{enc}}) \wedge (m, \delta) \in \mathcal{R}\}$.

The proof system $(\mathcal{P}, \mathcal{V})$ for the relation $\mathcal{R}_{\mathsf{proof\_system}}$ inherently satisfies the condition $(w, x) \in \mathcal{R}_{\mathsf{proof\_system}}$. With the completeness of this proof system, the probability of verifying x as true after generating a proof from $\mathcal{P}(\Psi, w, x)$ is 1:

$$Pr[(w, x) \in \mathcal{R}_{\mathsf{proof\_system}} \wedge 1 \leftarrow \mathcal{V}(\Psi, x)_{\mathcal{P}(\Psi, w, x)}] = 1$$

confirming the correctness of the protocol.

Soundness. The probability that an adversary can cheat the verifier into accepting a false statement is negligible. This is inferred from:

$$\begin{aligned} &Pr[(pk, sk) \leftarrow \mathsf{Gen}; (x, aux) \leftarrow \mathcal{A}(pk, sk, \Psi); b \leftarrow \mathcal{V}(\Psi, x); \\ m' \leftarrow \mathsf{Dec}_{sk}(C); m \leftarrow recon(pk, \Psi, \delta, m') &: b = 1 \wedge (m', \delta) \notin \mathcal{R}] \\ &< Pr[b \leftarrow \mathcal{V}(\Psi, x) : b = 1 \wedge (w, x) \notin \mathcal{R}_{\mathsf{proof\_system}}] = \mathsf{negl}(\lambda) \end{aligned}$$

The last equation comes from the soundness of the proof system.

Special honest-verifier zero knowledge. We follow the notation used in the proof of correctness. For $(w, x) \in \mathcal{R}_{\mathsf{proof\_system}}$, by CPA-security of the PKE scheme and the relation $\mathcal{R}$ is simulatable, the following real and random x distributions are indistinguishable:

- compute $(C, r_{\mathsf{enc}}) \leftarrow \mathsf{Enc}_{pk}(m)$ and $\delta = f(m)$, set $x = (C, pk, \delta)$
- randomly generate C, simulate $\delta \leftarrow \mathsf{Sim}(\lambda)$, set $x = (C, pk, \delta)$,

Furthermore, by special honest-verifier zero knowledge of the proof system, there exists a simulator Sim', such that the following α distributions are indistinguishable:

- Real transcript generation with private m as input: $(C, r_{\mathsf{enc}}) \leftarrow \mathsf{Enc}_{pk}(m), \delta = f(m)$, set $x = (C, pk, \delta)$, set $w = (m, r_{\mathsf{enc}}), \alpha \leftarrow Trans(\Psi, w, x)$
- Simulated transcript without knowing witness: randomly generate C, simulate $\delta \leftarrow \mathsf{Sim}(\lambda)$, set $x = (C, pk, \delta), \alpha \leftarrow \mathsf{Sim}'(\Psi, x)$

This property ensures that the verifier learns no additional information about the private witness beyond what is explicitly revealed in the statement x, gives the special honest-verifier zero knowledge of the constructed protocol.