Verifiable Decryption in the Head

Kristian Gjøsteen, Thomas Haines, Johannes Müller, Peter Rønne and Tjerand Silde
Presented at ACISP 2022

Improved Version:
Emil August Hovd Olaisen, Thomas Haines, Peter Rønne and Tjerand Silde
Presented at NIST Threshold Crypto Workshop 2024

February 14, 2025

Verifiable Decryption

For a ciphertext c and plaintext m we want a proof that

$$m = \mathrm{Dec}(c, sk)$$

Use cases include

- E-Voting: Decryption of individual votes or sum of votes.
- Decryption mix-nets for privacy
- Verifiable Functional Encryption
- Privacy-preserving financial applications

Often we need to decrypt many ciphertexts verifiably at the same time – we need something that scales well in this situation.

Motivation - JCJ

Juels, Catalano, Jakobsson 2005 (widely known as the JCJ scheme)

- Seminal paper on coercion-resistant voting protocols
- Not just receipt-freeness, but e.g. also security against
  - Randomization attacks
  - Forced abstention attacks
- *Efficiency problem:* Tally time quadratic in number of submitted ballots - needs a *Plaintext Equivalence Test* for each pair of ballots, and pair of voters and ballots.

Clarkson, Chong, Myers 2008, "Civitas"

- Implementation of JCJ
- Precinct-type ballot box design for better tally time, but less privacy
- Several specifications

JCJ - Main Idea

Verify eligibility of a ballot using a preregistrered credential.

- Voter authenticates ballot using her credential.
- To preserve privacy credential under encryption.
- Public verification of whether encrypted credential is in the list of registered credentials
- Done using *Plaintext Equivalence Test*

$$PET(\mathrm{Enc}(C_1), \mathrm{Enc}(C_2)) = \begin{cases} 1 & \text{if } C_1 = C_2 \\ \text{random} & \text{if } C_1 \neq C_2 \end{cases}$$

- Proof using homomorphic properties of encryption (ElGamal for JCJ)

$$(\mathrm{Enc}(C_1)/\mathrm{Enc}(C_2))^r = \mathrm{Enc}((C_1/C_2)^r)$$

- *Verifiable decryption* to reveal

$$(C_1/C_2)^r$$

Password-based Authentication

- Credentials can in principle be low entropy (does have problems for coercion resistance if entropy too low)

Could in principle to create a PAKE

- Alice and Bob exchange $\mathsf{pk}_A, \mathsf{pk}_B$
- Joint threshold PK from $\mathsf{pk} = \mathsf{pk}_A \star \mathsf{pk}_B$
- Alice and Bob submit $\mathsf{Enc}_{\mathsf{pk}_A}(\pi_a)$ and $\mathsf{Enc}_{\mathsf{pk}_B}(\pi_b)$ (plus ZKPs)
- Alice and Bob run (interactive) PET to test $\pi_a \stackrel{?}{=} \pi_b$ (test result can be kept secret or public)
- We can bind any key exchange to this check
- Can easily be generalised to a multi-user setting
- Very inefficient

Verifiable Decryption

Let us return to the problem of efficient verifiable decryption.

ElGamal Verifiable Decryption

Recall ElGamal Encryption

- Prime order cyclic group G
- Generator g
- Public key $\mathrm{pk} = g^x$, secret key $x = sk$
- $\mathrm{Enc}(m, r) = c = (c_1, c_2) = (g^r, \mathrm{pk}^r m) = (g^r, g^{rx} m)$
- Decrypt: $\mathrm{Dec}(c, x) = c_2 \cdot c_1^{-x} = m$

Given ciphertext c and plaintext m, to prove that $\mathrm{Dec}(c, x) = m$ it is enough to

- Publish $A = c_1^x$, Verifier should check $c_2 \cdot A^{-1} = m$
- Prove $\log_{c_1} A = \log_g \mathrm{pk}$
- Chaum-Pedersen proof of discrete log equality
- Efficient Sigma protocol
- Non-interactive via Fiat-Shamir transformation (or similar)

Post-quantum Verifiable Decryption

For many systems we can do efficient verifiable decryption.

For PQ encryption harder. There are schemes where we can compute randomness from ciphertext using the secret key (like Paillier), but this is not Zero-Knowledge proof decryption.

We get first efficient lattice-based verifiable decryption with only one server. Especially efficient when number of ciphertexts is much larger that the security parameter.

Really Stupid Idea

An easy way to verify any decryption would be to hand out the secret key and the verifier could decrypt herself.

The interesting case is where we want to preserve the secrecy of the decryption key, e.g. to keep secrecy of other ciphertexts.

Less Stupid Idea

Consider again ElGamal $\mathrm{pk} = g^x$ and $c = (c_1, c_2)$

- Split key $\mathrm{pk}_1 = g^{s_1}$ and $\mathrm{pk}_2 = g^{s_2}$. Verifier checks $\mathrm{pk} = \mathrm{pk}_1 \cdot \mathrm{pk}_2$. I.e. $x = s_1 + s_2$
- Decryption factorizes $A = c_1^x = c_1^{s_1} \cdot c_1^{s_2} = A_1 \cdot A_2$
- Prover gives $\mathrm{pk}_1, \mathrm{pk}_2, A_1, A_2$. Verifier checks $\mathrm{pk} = \mathrm{pk}_1 \cdot \mathrm{pk}_2$ and that $m = c_2 \cdot (A_1 \cdot A_2)^{-1}$
- Cut-and-choose: The verifier asks to see s_b for $b = 1, 2$ and checks $\mathrm{pk}_b = g^{s_b}$ and $A_b = c_1^{s_b}$
- Soundness $1/2$ (prover could use bad randomness in A_1 or A_2)

Can be repeated to have soundness error exponentially suppressed.

Less efficient than Chaum-Pedersen, but maybe it generalises to other schemes, especially with homomorphic properties?

General transformation - Idea

Verifiable Decryption from Distributed Decryption

A 2-party passively secure distributed decryption protocol can be transformed into a 1-party proof of correct decryption.

Like "MPC in the head":

- Even though we only have one player (the prover) we are going to artificially split into two players using the distributed decryption
- The verifier does a cut-and-choose between the two virtual players
- Prover reveals all secrets and randomness for the challenged virtual player
- The verifier checks that the key share and decryption shares are correctly constructed

General transformation - Advantages

Verifiable Decryption from Distributed Decryption

A 2-party passively secure distributed decryption protocol can be transformed into a 1-party proof of correct decryption.

Advantages:

- We get a soundness factor $1/2$ for each key distribution – but can be done at the same time for many ciphertexts decryptions
- Security (soundness error) level can be adapted dynamically in the interactive protocol, and kept low for risk-adverse adversaries
- Pre-computation possible for fast online phase
- Can be made non-interactive e.g. via Fiat-Shamir transformation
- We only need passively secure distributed decryption since we verify the honest construction of the decryption shares via the secret key and randomness

General transformation - Challenges

Verifiable Decryption from Distributed Decryption

A 2-party passively secure distributed decryption protocol can be transformed into a 1-party proof of correct decryption.

Some challenges

- We need to re-share the same decryption key – not just do a new key generation
- We need to be able to recover the secret key from the key shares for the security proof. I.e. need extra algorithm compared to standard distributed decryption
- We need to verify that the revealed key is correct (however not for each ciphertext)
- In security proof we need to simulate both key shares and the decryption shares

This means we need to change the definition of passively secure distributed decryption slightly compared to standard definitions.

Advantages

- Security (soundness error) level can be adapted dynamically
- Pre-computation possible for fast online phase
- Soundness error can be adapted e.g. to risk-adverse adversaries for more efficiency
- Fast for many ciphertexts decrypted at the same time

Passively secure distributed decryption

- Key generation algorithm `KeyGen`
- Encryption algorithm Enc
- Decryption algorithm Dec.
- Predicate KeyMatch with takes a public and secret key (New)

We require that for all matching public and secret keys pk, sk and message m, $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$.

Passively secure distributed decryption

A *distributed decryption protocol* for this public key cryptosystem consists of four algorithms (here for two parties)

- *The dealer algorithm* (Deal) takes as input a public key and corresponding secret key and outputs two *private key shares* and some *auxiliary data* (modified from standard definition).
- *The verify algorithm* (Verify) takes as input a public key, auxiliary data, an index and a secret key share and outputs *yes* (1) or *no* (0).
- *The player algorithm* (Play) takes as input a secret key share and a ciphertext and outputs a *decryption share*.
- *The reconstruction algorithm* (Reconstruct) takes as input a ciphertext and two decryption shares and outputs either $\perp$ or a message.
- *The find key algorithm* (FindKey) takes as input two secret key shares and returns a secret key (new).

Security of passively secure distributed decryption

Intuitively a distributed decryption protocol is *correct* if the Play and Reconstruct collectively recover the encrypted message and verification accepts when the dealer is honest.

Definition (Correctness)

A distributed decryption protocol is *correct* if for all ciphertexts c, any key pair $(\mathsf{pk}, \mathsf{sk})$ such that $\mathsf{KeyMatch}(\mathsf{pk}, \mathsf{sk}) = 1$, any $(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{aux})$ output by $\mathsf{Deal}(\mathsf{pk}, \mathsf{sk})$, then $\mathsf{Verify}(\mathsf{pk}, \mathsf{aux}, 0, \mathsf{sk}_0) = 1 = \mathsf{Verify}(\mathsf{pk}, \mathsf{aux}, 1, \mathsf{sk}_1)$ and

$$\Pr\left[m \leftarrow \mathsf{Dec}(\mathsf{sk}, c); \mathsf{Reconstruct}(c, \mathsf{Play}(\mathsf{sk}_0, c), \mathsf{Play}(\mathsf{sk}_1, c)) = m\right] \geq 1 - \mathsf{negl}$$

Security of passively secure distributed decryption

For a distributed decryption protocol we must trust the dealer for privacy, but not integrity.
Integrity ensures that if both secret shares given by the dealer are valid (according to the Verify algorithm) then the Play and Reconstruct will collectively recover the encrypted message.

Definition (Integrity)

A distributed decryption protocol has *integrity* if for all ciphertexts c, public keys pk, secret key shares $(\mathsf{sk}_1, \mathsf{sk}_2)$, and auxiliary data aux and sk output by $\mathsf{FindKey}(\mathsf{sk}_0, \mathsf{sk}_1)$ satisfying
$\mathsf{Verify}(\mathsf{pk}, \mathsf{aux}, 0, \mathsf{sk}_0) = 1 = \mathsf{Verify}(\mathsf{pk}, \mathsf{aux}, 1, \mathsf{sk}_1)$, we have that

$$\Pr\Big[\ \mathsf{KeyMatch}(\mathsf{pk}, \mathsf{sk}) \wedge \mathsf{Reconstruct}(c, \mathsf{Play}(\mathsf{sk}_0, c), \mathsf{Play}(\mathsf{sk}_1, c)) = \mathsf{Dec}(\mathsf{sk}, c)\Big] \geq 1 - \mathsf{negl}.$$

Security of passively secure distributed decryption

For standard threshold cryptosystems and distributed decryption, security is defined through the usual security games for public key cryptosystem, allowing the adversary access to the decryption key shares through decryption share oracles.
We need instead a variant of simulatability, namely we must be able to simulate both decryption key shares and decryption shares in a consistent fashion.

$Exp_{\mathcal{A}}^{ddp-sim-0}(\mathsf{pk}, \mathsf{sk})$

$(i, (c_0, ..., c_\tau), (m_0, ..., m_\tau)) \leftarrow_r \mathcal{A}(\mathsf{pk})$

$(\mathsf{sk}_0, \mathsf{sk}_1, \mathsf{aux}) \leftarrow_r \mathsf{Deal}(\mathsf{pk}, \mathsf{sk})$

$\forall j : \mathsf{ds}_j \leftarrow \mathsf{Play}(\mathsf{sk}_{1-i}, c_j)$

$b = \mathcal{A}(\mathsf{aux}, \mathsf{sk}_i, (\mathsf{ds}_0, ..., \mathsf{ds}_\tau))$

return b

$Exp_{\mathcal{A}}^{ddp-sim-1}(\mathsf{pk})$

$(i, (c_0, ..., c_\tau), (m_0, ..., m_\tau)) \leftarrow_r \mathcal{A}(\mathsf{pk})$

$(\mathsf{sk}_i, \mathsf{aux}) \leftarrow_r \mathsf{DealSim}(\mathsf{pk}, i)$

$\forall j : \mathsf{ds}_j \leftarrow \mathsf{PlaySim}(\mathsf{pk}, \mathsf{sk}_i, c_j, m_j)$

$b = \mathcal{A}(\mathsf{aux}, \mathsf{sk}_i, (\mathsf{ds}_0, ..., \mathsf{ds}_\tau))$

return b

Security of passively secure distributed decryption

Definition (Simulatability)

Consider a pair of algorithms DealSim and PlaySim and an adversary $\mathcal{A}$ playing the experiments from last slide, where $\mathcal{A}$ always outputs $\boldsymbol{c} = (c_0, ..., c_\tau), \boldsymbol{m} = (m_0, ..., m_\tau)$ such that $\{m_j = \mathsf{Dec}(\mathsf{sk}, c_j)\}_{j=1}^{\tau}$. The *simulatability advantage* of $\mathcal{A}$ is

$$Adv^{ddp-sim}(\mathcal{A}, \mathsf{pk}, \mathsf{sk}) = |\Pr[Exp_{\mathcal{A}}^{ddp-sim-0}(\mathsf{pk}, \mathsf{sk}) = 1] - \Pr[Exp_{\mathcal{A}}^{ddp-sim-1}(\mathsf{pk}) = 1]|,$$

where the probability is taken over the random tapes and $(\mathsf{pk}, \mathsf{sk})$ output by `KeyGen`. We say that a distributed decryption protocol is (t, ϵ)-*simulatable* (or just *simulatable*) if no t-time algorithm $\mathcal{A}$ has advantage greater than ϵ.

ElGamal Passively secure distributed decryption

- *The dealer algorithm* (Deal) takes as input a public key g^x and corresponding secret key x, samples x_0 from $\mathbb{Z}_p^*$, sets $x_1 = x - x_0$ and returns $(x_0, x_1, \text{aux} = (g^{x_0}, g^{x_1}))$.
- *The verify algorithm* (Verify) takes as input a public key pk, auxiliary data $\text{aux} = (\text{aux}_0, \text{aux}_1)$, an index i and a secret key share x and outputs 1 iff $(g^x = \text{aux}_i) \wedge (\text{pk} = \text{aux}_0\text{aux}_1)$.
- *The player algorithm* (Play) takes as input a secret key share x and a ciphertext (c_1, c_2) and outputs a *decryption share* c_1^x.
- *The reconstruction algorithm* (Reconstruct) takes as input a ciphertext c_1, c_2 and two decryption shares (t_0, t_1) and outputs $c_2/(t_0 t_1)$.
- *The find key algorithm* (FindKey) takes as input two key shares sk_0, sk_1 and outputs $\text{sk}_0 + \text{sk}_1$

ElGamal Passively secure distributed decryption

Privacy: The simulators DealSim and PlaySim work as follows:

- DealSim takes the public key pk and a bit i and samples x_i from $\mathbb{Z}_p^*$ and returns (wlog) $(x_i, (g^{x_i}, pk/g^{x_i}))$. It is clear that the auxiliary data and secret key from the simulator have the same distribution as the Deal.
- PlaySim takes as input public key pk, secret key x_i, ciphertext (c_1, c_2), and message m and returns a decryption share $c_2/(c_1^{x_i} m)$. Since m is the message encrypted in the ciphertext this is a perfect simulation if m is the correct decryption.

General Transformation

The passively secure distributed decryption, as defined above, allows us to make a Sigma protocol for correct decryption.

General Transformation

Π_{ZKPCD}

$\underline{\text{Prover}((\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau}), (\mathsf{sk}))}$		$\underline{\text{Verifier}(\mathsf{pk}, \{c_j\}_{j=1}^{\tau}, \{m_j\}_{j=1}^{\tau})}$
$k = 1, ..., \lambda$:		
$(\mathsf{sk}_{0,k}, \mathsf{sk}_{1,k}, \mathsf{aux}_k) \leftarrow \mathsf{Deal}(\mathsf{pk}, \mathsf{sk})$		
$i = 0, 1$:		
$j = 1, ..., \tau$:		
$t_{i,j,k} \leftarrow \mathsf{Play}(\mathsf{sk}_{i,k}, c_j; \rho_{i,k,j})$		
$w \leftarrow (\{\mathsf{aux}_k, \{t_{i,j,k}\}\})$		
	$\xrightarrow{w}$	
		$\boldsymbol{\beta} \xleftarrow{\$} \{0,1\}^{\lambda}$
	$\xleftarrow{\boldsymbol{\beta}}$	
$z \leftarrow (\{\mathsf{sk}_{\boldsymbol{\beta}[k],k}\}_k, \{\rho_{\boldsymbol{\beta}[k],k,j}\}_{k,j})$		
	$\xrightarrow{z}$	
		$k = 1, ..., \lambda$:
		$\mathsf{Verify}(\mathsf{pk}, \mathsf{aux}_k, \boldsymbol{\beta}[k], \mathsf{sk}_{\boldsymbol{\beta}[k],k}) \stackrel{?}{=} 1$
		$j = 1, ..., \tau$:
		$\mathsf{Play}(\mathsf{sk}_{\boldsymbol{\beta}[k],k}, c_j; \rho_{\boldsymbol{\beta}[k],k,j}) \stackrel{?}{=} t_{\boldsymbol{\beta}[k],j,k}$
		$\mathsf{Reconstruct}(c_j, t_{0,j,k}, t_{1,j,k}) \stackrel{?}{=} m_j$

Sigma Protocol

Completeness: Up to the possible negligible error introduced by decryption failures, completeness follows correctness of the distributed decryption protocol.

(Special) Soundness: By rewinding, any cheating prover with a significant success probability can be used to create two accepting conversations (w, β, z) and (w, β', z'), with $\beta \neq \beta'$. From this it follows that for at least one k, $\beta[k] \neq \beta'[k]$, the verify algorithm has accepted both secret key shares and every decryption share in this round has been correctly created using the Play algorithm. Then, since the ciphertexts are encryptions of the first message vector, integrity implies that FindKey will recover a witness which matches the public key and for which the messages match the output of the decryption function.

Honest-Verifier Zero-Knowledge: Here we build a simulator using DealSim and PlaySim.

Machine-Checked Proof

Transformation checked in Coq

- Assuming perfect correctness, integrity and simulatability
- Instantiated for ElGamal
- Does not work for lattice-based primitives (presently)

Lattice-based verifiable decryption

The main novelty is that we can construct a passively secure distributed decryption for lattice-based encryption and use this to get decryption proofs. We use

- BGV encryption by Brakerski, Gentry and Vaikuntanathan
- BGV encryption is also used as commitment scheme

Lattice-based verifiable decryption

BGV Setup and Key generation

- $p \ll q$ be primes, let R_q and R_p be polynomial rings
- Key Generation: Sample $a \leftarrow\$ R_q$ and sample short $s, e \leftarrow\$ R_q$ such that $\max(\|s\|_\infty, \|e\|_\infty) \leq B_\infty$
- $\mathtt{pk} = (a, b) = (a, as + pe)$ and the secret key $\mathtt{sk} = (s, e)$

BGV Encryption of m in R_p

- Sample short $r, e', e'' \leftarrow\$ R_q$
- $\mathrm{Enc}_{\mathrm{pk}}(m) = (u, v) = (ar + pe', br + pe'' + m)$

BGV Decryption

- $m = (v - su \mod q) \mod p$

Distributed decryption of $c = (u, v)$ (building on Bendlin and Damgård)

- $s = s_1 + s_2 + ... + s_\xi$ secret key shares
- Party j computes $m_j = s_j u$
- sample some large noise $E_j \leftarrow\$ \mathbb{E} \subset R_q$, $\|E_j\|_\infty \leq 2^{\mathtt{sec}}(B_{\mathtt{Dec}}/p\xi)$
- $\mathrm{ds}_j = m_j + pE_j$
- $m \equiv (v - (\mathrm{ds}_1 + ... + \mathrm{ds}_\xi) \mod q) \mod p$

Lattice-based verifiable decryption

- Exact Amortized Zero-Knowledge Proof of Short Openings. This is the main novelty compared to ElGamal. These proofs are needed in the deal algorithm to prove that the decryption keys have short randomness. The proofs need to be verified by the Verify-algorithm.
- More efficient than previous proposals and comparable proof size to simultaneously developed proof technique by Lyubashevsky et al. (PKC'21)

Improved version

Use almost linear decryption from Boyle et al (EuroCrypt 2019)
We use

- 440× smaller proof size
- 10× smaller proof size than Lyubashevsky et al.

New Idea

- Before verify either gets s_0 or s_1
- New idea: Use homomorphic structure and let verifier challenge with a, b $(a \neq b)$ and get $as_0 + bs_1$
- For lattice-based encryption a, b needs to be small and error needs to be adapted