D–Shuffle for Prêt à Voter

Dalia Khader

D–Shuffle for Prêt à Voter

Dalia Khader
daliakhader@googlemail.com

Abstract. Prêt à Voter is an end–to–end verifiable voting scheme, that uses paper based ballot forms that are turned into encrypted receipts. The scheme was designed to be flexible, secure and to offer voters a familiar and easy voting experience. Secrecy of the vote in Prêt à Voter relies on encoding the vote using a randomized candidate list in the ballots. In a few variants of Prêt à Voter a verifiable shuffle was used in the ballot generation phase in order to randomize the candidates. Verifiable shuffles are cryptographic primitives that re–encrypt and permute a list of ciphertexts. They provide proofs of correctness of the shuffle and preserve secrecy of the permutation. This paper proposes a new verifiable shuffle "D–Shuffle" that is efficient. We provide a security proof for the D–Shuffle. Furthermore, we show that using the D–shuffle for generating ballots in Prêt à Voter scheme ensures its security against: "Authority Knowledge Attack" and "Chain of Custody Attack".
Keywords: E-voting, Verifiable Shuffle, Zero knowledge proofs.

1 Introduction

A shuffle is a permutation and re-randomization of a set of ciphertexts. Shuffling itself is relatively easy; the challenge is to provide a proof of correctness of a shuffle that anyone can verify without revealing the permutation. A mix-net is a series of chained servers each of which applies a shuffle to some input ciphertexts, before passing the output to the next server. Mix-nets were used widely in e-voting schemes. The main motivation in using them is to submit encrypted votes into a mix-net where every mix-server shuffles the votes. The output of the mix-net is then decrypted providing anonymity to the voters. Using verifiable shuffles prevents mix-servers from cheating.

In this paper we focus on one of the well known end-to-end verifiable schemes: Prêt à Voter . The Prêt à Voter approach to verifiable voting, randomizing candidate order on ballot to encode votes, was first proposed by Ryan in [16]. Since then several papers were introduced to add extra interesting properties to the original scheme [16, 18, 21]. Verifiable shuffles were used in Prêt à Voter either to mix the encrypted receipts before publishing them on a public bulletin board, and/or to randomize the candidates on the ballot. The shuffle we propose in this paper is focused on the latter case.

1.1 Prêt à Voter Overview

In "Prêt à Voter " ballots are given to voters via a confidential channel. The ballot has a left hand side (LHS) with a randomly permuted list of candidates,

and a right hand side (RHS) which carries an encryption of the order of the candidates in the LHS, usually referred to as *the onion* for historical reasons. Each ballot has a unique serial number (which could be a hash of the onion), (SN), for administrative purposes such as searching for the ballot on the bulletin board, etc (See Figure 3, Original scheme).

The voting takes place in the polling station. In the booth, the voter places a mark next to the name of the candidate she wants to vote for. She separates the RHS from LHS, shreds the LHS and takes the RHS to an official who scans and sends it to the tallying authority. A signed copy of the RHS is given to the voter to keep. The onions are used in the tabulation to interpret the voter's mark on the scanned RHS, enabling the tallying authorities to count the votes. The voter can verify that her vote has been received by checking the onion, serial number and choice of index, against the published results on the bulletin board.

The details of the procedure of tabulation, randomization of ballots, tallying, distributing the ballots, etc, varies in the different versions of Prêt à Voter [16, 18, 21]. On a conceptual level the procedure is the same. Random auditing of the ballots is used in all versions of Prêt à Voter to ensure the well-formedness of ballot forms. The auditing procedure involves decrypting onions on selected ballot forms and checking that they correspond to the LHS order. Given that the authorities responsible of creating the ballots can not predict which ballots will be chosen for auditing, it is hard to cheat without a high possibility of getting caught.

1.2 Motivation and Contribution

The Victorian State elections [5, 4, 3] considered developing the first state government-level universally verifiable public e-voting system in the world, based on Prêt à Voter . The proposed mechanism of constructing the ballot was adopted from [21] and is based on using a verifiable shuffle to permute the candidates on the ballot. The proof of shuffle is used as proof of well formness of the ballot. The scheme in [21] had two vulnerabilities:

- Authority knowledge attack: All ballots are generated by one authority. Therefore this authority is trusted to maintain both privacy and receipt-freeness. Generating the ballots in a distributed fashion is desirable, because it ensures that no one but the voter ever learns the candidate ordering. However, there are three major obstacles preventing from that in [21]:
  - Proving the ballot is well-formed in the distributed fashion.
  - Printing the ballot without the printer(s) learning the order.
  - Ensuring robustness so that the scheme can be run even in the presence of some dishonest election officials.
- Chain of Custody: The ballot secrecy in Prêt à Voter relies on the fact that no one can know the order of the candidates unless they own the decryption key of the onion. However, the ballot form LHS contains the candidate order as plaintext. This means that the chain of custody between the ballot generation and until the ballot reaches the voter should be trusted. Ryan and Peacock

have discussed an alternative approach [17] referred to as Print–on–Demand. The idea is to print ballot forms at the point they are needed. The ballot will have two onions–the LHS one which can be decrypted in the polling station, and RHS one which can be decrypted by the Prêt à Voter tellers as in the original scheme.

The verifiable shuffle needs to be efficient to cope with the number of ballots generated and verified in the election time. In this paper we propose an efficient and secure verifiable shuffle for that purpose referred to as the D–shuffle (it uses disjunctive proofs for verifying the shuffle hence the name). The D–shuffle can also provide a distributed way of creating the ballot and can provide parallel shuffling that enables Print–on–Demand with minimum computational cost.

2 The Design of the Verifiable D–Shuffle

In the design of the D–Shuffle we require an encryption scheme with Homomorphism and Re-encryption properties. Assume we have an encryption scheme $E = (KeyGen, Enc, Dec)$. Let the key pair generated be (pk, sk). Let r_1, r_2 be the randomization factors used in encrypting. Let M, M_1, M_2 be plaintext messages. The properties we require in this paper are:

- Homomorphism: Multiplying two ciphertexts results with a third ciphertext such that: $Enc(pk, M_1, r_1).Enc(pk, M_2, r_2) = Enc(pk, M_1 + M_2, r_1 + r_2)$;
- Re-encryption: An encryption $CT = Enc(pk, M, r_1)$ can be re-encrypted such that $ReEnc(CT, r_2) = Enc(pk, M, r_1 + r_2)$.

The general idea behind our shuffle is derived from Theorem 1. We explain the Theorem using Definition 1 and prove it as follows;

Definition 1. *Sequence $M = (m_1, \ldots, m_n)$ is a super–increasing sequence if every element of the sequence is positive integers and is greater than the sum of all previous elements in the sequence (i.e. $m_k > \sum_{i=1}^{k-1} m_i$).*

Theorem 1. *Let $M = (m_1, \ldots, m_n)$ be a super–increasing sequence and $S = \sum_{i=1}^{n} m_i$. If $X = (x_1, \ldots, x_n)$ is a solution of*

$$S = \sum_{i=1}^{n} x_i$$

such that $\forall j \in \{1, \ldots, n\} : x_j \in M$, then $(x_1, \ldots, x_n)$ is a permutation of M.

Proof. Recall the subset sum problem: a sequence of integers M and an integer S, find any non-empty subset X that sums to S. This problem is proven to have either one unique solution or none [11] over super-increasing sequences. Given Theorem 1 assumes the existence of the subset $X \subseteq M$ and assumes that $S = \sum_{i=1}^{n} x_n$ then by the uniqueness property X is a permutation of M.

2.1 Intuition Behind the Design

We explain the intuition behind our design of the D–Shuffle using Theorem 1. The general idea is to assume all elements of Theorem 1 are encrypted and we prove the theorem holds using zero knowledge proofs. Let M be a super–increasing sequence that is encrypted and fed to the D–Shuffle as input. Assume the output is the encrypted version of $X = (x_1, \ldots, x_n)$. According to Theorem 1 the output is a permutation if the following two conditions hold:

1. **The Belonging Condition:** $\forall j \in \{1, \ldots, n\} : x_j \in M$.
   In the D–Shuffle this is equivalent to saying "All output ciphertexts belong to the list of all input ciphertexts". We require the disjunctive re-encryption proof shown in Figure 1.
2. **The Summation Condition:** $S = \sum_{i=1}^{n} m_i = \sum_{i=1}^{n} x_k$.
   In the D–Shuffle this is equivalent to saying that the homomorphic summation of the input ciphertexts and the homomorphic summation of the output ciphertexts are encryptions of the same plaintext value (i.e. the output sum is just a re–encryption of the input sum). We require the re-encryption proof shown in Figure 2.

Statement: Given the ciphertext $\grave{c}_j$ and list of ciphertexts $\{c_1, \ldots, c_n\}$ prove the knowledge of r such that the following is true: $[\grave{c}_j = ReEnc(c_i, r)] \wedge [c_i \in \{c_1, \ldots, c_n\}]$
Creating the proof:
$\pi_j = DRE.Proof(\{c_1, \ldots, c_n\}, \grave{c}_j, pk, r)$
Verifying:
$\{0, 1\} = DRE.Verify(\{c_1, \ldots, c_n\}, \grave{c}_j, pk, \pi_j)$

Fig. 1: Disjunctive Re-Encryption (DRE)

Statement: Given two ciphertexts $c, \grave{c}$ prove knowledge of r such that: $\grave{c} = ReEnc(c, r)$
Creating the proof: $\pi = RE.Proof(c, \grave{c}, r)$
Verifying: $\{0, 1\} = RE.Verify(c, \grave{c}, \pi)$

Fig. 2: Re-Encryption Zero Knowledge Proof (RE)

2.2 The Construction of the D–Shuffle

Let the plaintext we intend to encrypt and shuffle be the super–increasing sequence $M = (m_1, \ldots, m_n)$. We start with creating a list $\{c_1, \ldots, c_n\}$ such that $c_k = Enc(pk, m_k, 1)$. Note that we can verify correctness of the encryption easily since the randomization is 1.

The Shuffling Procedure:

1. Choose $r_1 \ldots r_n$ random values.

2. Create the output list $\{\grave{c}_1, \ldots, \grave{c}_n\}$ of ciphertexts by re–encrypting and permuting such that $\grave{c}_j = ReEnc(c_k, r_k)$ for some $c_k \in \{c_1, \ldots, c_n\}$.
3. Create $\pi_j = DRE.Proof(\{c_1, \ldots, c_n\}, \grave{c}_j, pk, r)$.
4. Let $S = \sum_{k=1}^{n} m_k$ and $R = \sum_{k=1}^{n} r_k$.
5. Let $C = Enc(pk, S, n+1) = \prod_{k=1}^{n} c_k$. Note that the randomization factor equals $n+1$ since the randomization factors of the c_k is all equal to 1.
6. Let $\grave{C} = \prod_{k=1}^{n} \grave{c}_k = Enc(pk, S, n+1+R) = ReEnc(C, R)$.
7. Create Re-Encryption Zero Knowledge Proof $\bar{\pi} = RE.Proof(C, \grave{C}, R)$.

One can have a mix–net where each mix–server i runs the D–shuffle on the output of the server $i-1$.

Verifying the Shuffle:

1. Compute from the input ciphertexts $C = \prod_{k=1}^{n} c_k$.
2. Compute from the output ciphertexts $\grave{C} = \prod_{k=1}^{n} \grave{c}_k$.
3. For all j s.t. $j \in \{1, \ldots, n\}$; Check $DRE.Verify(\{c_1, \ldots, c_n\}, \grave{c}_j, pk, \pi_j) = 1$.
4. Check $RE.Verify(C, \grave{C}, \bar{\pi}) = 1$.

2.3 Security of the D–Shuffle

There are three properties that a verifiable shuffle should achieve: secrecy of the permutation, soundness of the proofs, and correctness of the proofs.

- Correctness of the verification of the shuffle implies that an honest prover (shuffler) has to be able to create the re-encryption and zero knowledge proofs such that they verify correctly. This is achieved by assuming correctness of the zero knowledge proofs used in Figure 1 and Figure 2.
- Soundness of the verification of shuffle implies that no dishonest prover (shuffler) can produce a proof of shuffle that verifies correctly. This is guaranteed with the soundness of the proofs and the uniqueness property in Theorem 1.
- Secrecy of the permutation depends on two security notions, the zero knowledge property of the proofs and on the security of the encryption scheme (see appendix, IND-V-CPA,IND-V-CCA1, or IND-V-CCA2).

2.4 On Instantiations of the D–shuffle

The D–Shuffle requires homomorphic properties to verify the sums. Exponential ElGamal and Paillier were heavily used for voting applications for their homomorphic properties. In the Victorian State elections [5, 4, 3] the suggestion was

to use Exponential ElGamal. In this paper we focus on having general constructions of the D–shuffle and Prêt à Voter . Recent security analysis showed that using IND–CCA2 encryptions for creating ballots [1, 2, 9] is sufficient to guarantee secrecy of vote. If we require both homomorphic properties and IND–CCA2 security then we can use Naor–Yung encryptions [1, 2, 9] or Cramer–Shoup encryption [13, 20]. The two mentioned encryption schemes have an extractable part of the cipher that is homomorphic.

2.5 On the Efficiency of D–Shuffle

The main advantage of the D–Shuffle is the fact that it is non–interactive. The first non-interactive verifiable shuffle was proposed in [7], however, the proofs were extremely large $15n + 120$, where n is the number of ciphertexts being shuffled. In a more recent result by Lipmaa and Zhang [12], the size of the proof dropped to $6n+11$. In the D–shuffle assuming we use ElGamal Exponential, the disjunctive zero knowledge proof for an ElGamal encryption is $2n$ and the zero knowledge proof of Re-Encryption for ElGamal is two more elements, causing the total to drop to $2n + 2$. Jakobsson et al [8] proposed a technique for making mix nets robust and efficient, called randomized partial checking. The general idea is to ask each server to reveal a pseudo-randomly selected subset of its input/output relations, therefore providing strong evidence of correctness. The secrecy of the permutation also gets compromised using such a technique to a certain level [10]. The D–shuffle allows the verifier to choose the balance between "correctness proofs vs efficiency" as they require without compromising secrecy. The verifier can choose randomly the number of disjunctive proofs he would like to obtain since the proofs are independent and given the disjunctive proofs do not reveal any input/output relations, the permutation remains secret.

3 The D-Shuffle used for Prêt à Voter

The ceremony of the voting, tabulation and verification of the vote remain unchanged as described in §1.1. Each candidate is presented in a code m_i such that the set $(m_1, \ldots, m_n)$ is super–increasing and is publicly announced. The onion contains an encrypted list of the different candidates i.e. a permutation of $\{\mathsf{Enc}(pk, m_1, r_1), \mathsf{Enc}(pk, m_2, r_2), \ldots, \mathsf{Enc}(pk, m_k, r_k)\}$ that corresponds to the order of the candidates on the LHS.

The ballot creation uses the D–Shuffle such that each mix–server i verifies the zero knowledge proofs of server $i-1$, shuffles the outputs of $i-1$ and publishes the new zero knowledge proofs on a bulletin board. The initial ciphertexts input to mix–net, i.e. server $i = 1$, is $\{\mathsf{Enc}(pk, m_1, 1), \ldots, \mathsf{Enc}(pk, m_k, 1)\}$ which is verifiable by everyone since the randomization factors are 1 and the candidate codes is public information. The final list of ciphertexts is printed as the onion on the ballot. The auditing of the ballot and checking its well formness can be done in three ways depending on level:

– Extreme Auditing: Verifying all the zero knowledge proofs on the bulletin board. This can be done by any entity that has the means and computational powers. Partial checking of the proofs can be applied here.
– Basic Auditing: Checking the onion on the ballots against the Serial Number SN on the bulletin board. This can be by any entity that is willing to act as an observer to the elections and no computational power or cryptographic knowledge is required.
– Voter Auditing: Decrypting the RHS and checking it against the LHS. This is the traditional Prêt à Voter technique used by the voters to audit the ballots if they want too.

Among the three techniques, the voter auditing technique is the most user friendly for the voters. However, ballots used for auditing using that technique should not be used for voting. This can be enforced because the only way to audit is to ask the authorities with the decryption key to reveal the candidates order in the onion and at that point the SN is flagged as unusable for election.

Multi-Authority Ballot Generation Each mix-server in the mix-net can be considered an independent authority such that the values that correspond to the ciphertexts published on the final ballot are unknown to any of them. Therefore the privacy and receipt-freeness can not be broken by any of the mix-servers or any number of them. To break privacy and receipt-freeness all mix-servers have to collude. This partially solves "Authority knowledge attack":

– Proving the ballot is well-formed in the distributed fashion. Each mix-server publishes enough zero knowledge proofs to verify that the shuffling is correct and honest. Therefore the final printed ballot is proven well formed given all the proofs published verify correctly.
– Ensuring robustness so that the scheme can be run even in the presence of some dishonest election officials. This is done using the three auditing techniques mentioned earlier.

Print–On–Demand vs Preprinted Ballots In Prêt à Voter secrecy of the ballot relies on the assumption that the LHS was not revealed to any entity other than the voter. This means that the chain of custody between the creation of a ballot form and its use in a polling station needs to be trusted. Alternatively, the ballot can be printed in the polling station at the time of the vote [19]. This is what is referred to as print–on–demand scenario. The ballot given to the voter will have two onions one that can be decrypted in the booth in private and printed out to resemble the LHS and the other is the traditional Prêt à Voter onion existing on the RHS (See Figure 3) which is decrypted in the tallying phase. This avoids the chain of custody issues. Ballot forms can be audited in the same way as previously, by printing the RHS first, and then checking that it matches the LHS. The voting experience with the exception of printing the LHS remains the same too. To achieve print–on–demand one can provide a double

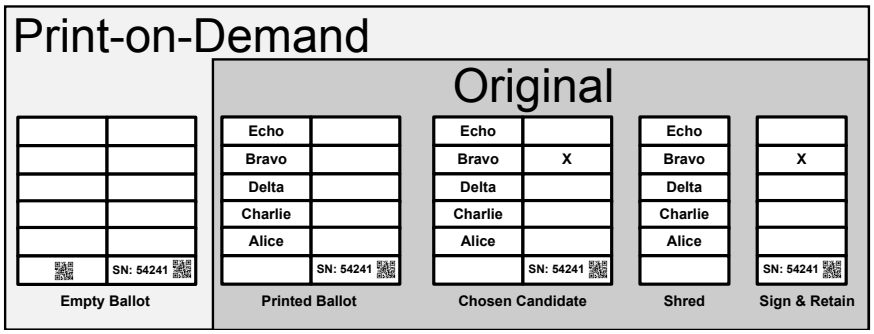

Fig. 3: Prêt à Voter : The Ceremony

Statement: Given the pair of ciphertext $(c\grave{1}_{,j}, c\grave{2}_{,j})$ and list of pair of ciphertexts $\{(c_{1,1}, c_{2,1}), \ldots, (c_{1,n}, c_{2,n})\}$ prove the knowledge of r_1, r_2 such that the following is true: $[c\grave{1}_{,j} = ReEnc(c_{1,i}, r_1)] \wedge [c\grave{2}_{,j} = ReEnc(c_{2,i}, r_2)] \wedge [(c_{1,i}, c_{2,i}) \in \{(c_{1,1}, c_{2,1}), \ldots, (c_{1,n}, c_{2,n})\}]$
Creating the proof:
$\pi_j = DDRE.Proof(\{(c_{1,1}, c_{2,1}), \ldots, (c_{1,n}, c_{2,n})\}, (c\grave{1}_{,j}, c\grave{2}_{,j}), pk, r_1, r_2)$
Verifying:
$\{0,1\} = DDRE.Verify(\{(c_{1,1}, c_{2,1}), \ldots, (c_{1,n}, c_{2,n})\}, (c\grave{1}_{,j}, c\grave{2}_{,j}), pk, \pi_j)$

Fig. 4: Disjunctive Double Re-Encryption

ciphertext disjunctive zero knowledge proof as shown in the Figure 4 in place of the disjunctive proofs used earlier.

The print-on-demand solves the remaining two problems:

- Chain of Custody: The ballots are generated such that parallel shuffling takes place and no mix server knows the final order. The ballot generated does not contain any plaintext and the LHS is encrypted from the point the ballot is generated and until it reaches the polling station.
- Authority knowledge attack regarding printers: The double ciphered mix-net can be implemented such that we have multiple printers in the booth. Assume we have three printers in each booth, then we can replace the key pair(pk_L, sk_L) with (pk_{p1}, sk_{p1}), (pk_{p2}, sk_{p2}) and (pk_{p3}, sk_{p3}). Each printer outputs part of the LHS and none of the printers will fully know the ballot.

4 Conclusion

We propose a new verifiable shuffle referred to as the D–Shuffle. The new shuffle is efficient, sound, complete, and ofcourse reserves the secrecy of the permutation. The D–shuffle when used for creating ballots in a Prêt à Voter scheme, it prevents "Authority Knowledge Attack" and "Chain of Custody Attack".

References

1. D. Bernhard, V. Cortier, O. Pereira, B. Smyth, and B. Warinschi. Adapting helios for provable ballot privacy. In *ESORICS'11*, pages 335–354. LNCS.
2. D. Bernhard, O. Pereira, and B. Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *ASIACRYPT*, pages 626–643. LNCS, 2012.
3. R. Buckland and R. Wen. The future of e-voting in australia. *IEEE Security& Privacy*, 10(5):25–32, 2012.
4. C. Burton, C. Culnane, J. Heather, T. Peacock, P. Y. A. Ryan, S. Schneider, S. Srinivasan, V. Teague, R. Wen, and Z. Xia. A supervised verifiable voting protocol for the victorian electoral commission. In *E-Voting'12*, pages 81–94.
5. C. Burton, C. Culnane, J. Heather, T. Peacock, P. Y. A. Ryan, S. Schneider, S. Srinivasan, V. Teague, R. Wen, and Z. Xia. Using prêt à voter in victorian state elections. In *EVT/WOTE'12*. USENIX, 2012.
6. D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer.
7. J. Groth and S. Lu. A non-interactive shuffle with pairing based verifiability. In *ASIACRYPT, LNCS series*, pages 51–67, 2007.
8. M. Jakobsson, A Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *EVT/WOTE*. USENIX, 2002.
9. D. Khader and P. Y. A. Ryan. Receipt freeness of prêt à voter provably secure. *IACR Cryptology ePrint Archive*, page 594, 2011.
10. S. Khazaei and D. Wikström. Randomized partial checking revisited. In *CT-RSA*, pages 115–128. LNCS, 2013.
11. Koblitz. *A course in number theory and cryptography*. Springer-Verlag, 1987.
12. H. Lipmaa and B. Zhang. A more efficient computationally sound non-interactive zero-knowledge shuffle argument. In *Security and Cryptography for Networks*, pages 477–502, 2012.
13. J Loftus, A. May, N P. Smart, and F. Vercauteren. On cca-secure somewhat homomorphic encryption. In *SAC'11*, pages 55–72. LNCS.
14. H. Martin and K Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT'00*, pages 539–556. LNCS.
15. T. P. Pedersen. A Threshold Cryptosystem without a Trusted Party. In *EUROCRYPT*, number 547 in LNCS, pages 522–526. Springer, 1991.
16. P. Y. A. Ryan. A variant of the chaum voter-verifiable scheme. In *Issues in the theory of security*, WITS, pages 81–88. ACM, 2005.
17. P. Y. A. Ryan and T. Peacock. Threat analysis of cryptographic election schemes., 2006. CS-TR:971 NCL.
18. P.Y.A Ryan, D. Bismark, J. Heather, S. Schneider, and Zhe Xia. Prêt à voter: a voter-verifiable voting system. *Trans. Info. For. Sec.*, 4:662–673, 2009.
19. P.Y.A. Ryan and T. Peacock. Prêt à voter: a systems perspective. CS.TR.929, 2005. NCL.
20. D. Wikström. A universally composable mix-net. In *TCC*, pages 317–335. LNCS, 2004.
21. Z. Xia, C. Culnane, J. Heather, H. Jonker, P. Y. A. Ryan, S. Schneider, and S. Srinivasan. Versatile Prêt à Voter : Handling Multiple Election Methods with a Unified Interface. In *INDOCRYPT*, volume 6498, pages 98–114. LNCS, 2010.

A Secrecy of the D–Shuffle

We recall the definition of IND-CCA2, given a public-key encryption scheme that consists of the three algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.

Definition 2. *A public-key encryption scheme achieves IND-CCA2 security if any polynomial time attacker only has negligible advantage in the attack game, shown in Fig. 5. Note that the advantage is defined to be* $|\Pr[b' = b] - \frac{1}{2}|$.

1. *Setup.* The challenger takes the security parameter λ as input, and runs KeyGen to generate (pk, sk).
2. *Phase 1.* The attacker is given pk and can issue a polynomial number of decryption queries with any input: Given C, the challenger returns $\mathsf{Dec}(C, sk)$. At some point, the attacker chooses M_0, M_1 of equal length and sends them to the challenger for a challenge.
3. *Challenge.* The challenger selects $b \in_R \{0, 1\}$ and returns $C_b = \mathsf{Enc}(M_b, pk)$ as the challenge.
4. *Phase 2.* The attacker can issue a polynomial number of decryption oracle queries with any input except for C_b.
5. *Guess*: At some point the attacker terminates Phase 2 by outputting a guess b' for b.

Fig. 5: IND-CCA2 Game

In Definition 2, if we remove *Phase 2* in the attack game then it becomes the definition for IND-CCA1. Furthermore, if we completely disallow the attacker to access the decryption oracle then it becomes the standard IND-CPA security.

A.1 Indistinguishable Vectors of Ciphertexts

To facilitate our security analysis of the D–Shuffle, we proposed a different security model (i.e. IND-V-CCA2 security) for public key encryption schemes. We show that this new security model is equivalent to the standard IND-CCA2.

Definition 3. *A public-key encryption scheme achieves IND-V-CCA2 security if any polynomial time attacker only has negligible advantage in the attack game, shown in Fig. 6.*

In the model in Figure 6, if we remove *Phase 2* in the attack game then it becomes the definition for IND-V-CCA1 (equivalent to IND-CCA1). Furthermore, if we completely disallow the attacker to access the decryption oracle then it becomes the standard IND-V-CPA (equivalent to IND-CPA) security.
We proof the following theorem:

Theorem 2. *If there exist an Adversary* $\mathcal{A}^{\dagger}$ *that breaks the IND-V-CCA2 then their exist an Adversary* $\mathcal{A}$ *that can break the IND-CCA2 (See Figure 7)*

Note that if $b = 1$ the simulation is unfaithful, however the probability of guessing the right $\grave{b}$ remains. Adding up the probability of winning when $b = 0$ leads to $\frac{\epsilon}{4} + \frac{1}{4}$ and probability of winning when $b = 1$ is $\frac{1}{4}$. Advantage of winning is: $Adv_{IND-CCA2}(k) = |Pr[\mathcal{A} \text{ winning}] - \frac{1}{2}| = \frac{\epsilon}{4}$. This advantage is non-negligible when ϵ is non-negligible.

1. *Setup.* The challenger takes the security parameter λ as input, and runs KeyGen to generate (pk, sk).
2. *Phase 1.* The attacker is given pk and can issue a polynomial number of decryption queries with any input: Given C, the challenger returns $\mathsf{Dec}(C, sk)$. At some point, the attacker chooses a list $M_1, \ldots, M_n$ of equal length and two permutation P_0, P_1 and sends them to the challenger for a challenge.
3. *Challenge.* The challenger computes $\forall k \in \{1, \ldots, n\}; C_k = \mathsf{Enc}(M_k, pk)$. The challenger computes them according to P_0, P_1 such that:
   $\{\grave{C}_1, \ldots, \grave{C}_n\} = P_0(\{C_1, \ldots, C_n\})$
   $\{\tilde{C}_1, \ldots, \tilde{C}_n\} = P_1(\{C_1, \ldots, C_n\})$.
   The challenger sets $E_0 = (\{\grave{C}_1, \ldots, \grave{C}_n\}, \{\tilde{C}_1, \ldots, \tilde{C}_n\})$ and $E_1 = (\{\tilde{C}_1, \ldots, \tilde{C}_n\}, \{\grave{C}_1, \ldots, \grave{C}_n\})$. The challenger randomly chooses $b \in \{0, 1\}$, and sends E_b to adversary.
4. *Phase 2.* The attacker can issue a polynomial number of decryption oracle queries with any input except for $C \notin E_b$.
5. *Guess*: At some point the attacker terminates Phase 2 by outputting a guess b' for b.

Fig. 6: IND-V-CCA2 Game

1. *Setup.* The challenger takes the security parameter λ as input, and runs KeyGen to generate (pk, sk). He gives the public parameters to $\mathcal{A}$ who forwards them to $\mathcal{A}^\dagger$
2. *Phase 1.* Every time $\mathcal{A}$ queries the decryption oracle from $\mathcal{A}^\dagger$, $\mathcal{A}^\dagger$ queries the decryption oracle from the challenger. The response of the challenger is forwarded to $\mathcal{A}$.
3. *Challenge.* The $\mathcal{A}^\dagger$ sends $M_0, \ldots, M_n$ together with two permutations (P_0, P_1), to $\mathcal{A}$. $\mathcal{A}$ computes the ciphertexts $C_1, \ldots, C_n$ such that $C_i = Enc(M_i, pk)$. The $\mathcal{A}$ forwards M_0, M_1 to the challenger. The challenger selects $b \in_R \{0, 1\}$ and returns $C_b = \mathsf{Enc}(M_b, pk)$ as the challenge. $\mathcal{A}$ assigns $C_0 = C_b$ and then permutes: $\{\grave{C}_1, \ldots, \grave{C}_n\} = P_0(\{C_1, \ldots, C_n\})$; $\{\tilde{C}_1, \ldots, \tilde{C}_n\} = P_1(\{C_1, \ldots, C_n\})$; $\mathcal{A}$ sets: $E_0 = (\{\grave{C}_1, \ldots, \grave{C}_n\}, \{\tilde{C}_1, \ldots, \tilde{C}_n\})$; $E_1 = (\{\tilde{C}_1, \ldots, \tilde{C}_n\}, \{\grave{C}_1, \ldots, \grave{C}_n\})$; Finally he flips a coin d and sends E_d to $\mathcal{A}$
4. *Phase 2.* Querying the decryption oracle is constraint to not sending the ciphertexts of the challenge.
5. *Guess*: $\mathcal{A}^\dagger$ returns a guess $\grave{d}$. If $\grave{d} = d$, the adversary $\mathcal{A}$ guesses $\grave{b} = 0$ else flip a coin to decide on $\grave{b}$

Fig. 7: IND-V-CCA2 versus IND-CCA2

A.2 On the Secrecy of the D–Shuffle Permutation

The two properties of correctness and soundness derive directly from the Zero Knowledge proofs. In this section we elaborate more on the secrecy of the permutation. Imagine their exist an adversary $\mathcal{A}$ that can guess the permutation of the verifiable D–Shuffle. This adversary can be used as a subroutine for $\mathcal{A}^\dagger$ for breaking the IND-V-(CPA,CCA1,CCA2) as follows:

– In the challenge, the adversary $\mathcal{A}^{\dagger}$ chooses $M_1, \ldots, M_n$, P_0, and P_1 sends them to $\mathcal{C}$.
– He receives back E_b back. Note that E_b has two permuted lists.
– $\mathcal{A}^{\dagger}$ chooses the first list and simulates the zero knowledge proofs.
– $\mathcal{A}^{\dagger}$ sends the list and the simulated proofs to $\mathcal{A}$.
– $\mathcal{A}$ should return either P_0 or P_1. If P_0 is returned then $\mathcal{A}^{\dagger}$ answers back his guess as $\grave{b} = 0$ otherwise $\grave{b} = 1$

Furthermore, the same adversary $\mathcal{A}$ can be used as a sub–routine for $\mathcal{A}^{\ddagger}$ to break the zero knowledge properties as follows:

– $\mathcal{A}^{\ddagger}$ encrypts a list of $M_1 \ldots, M_n$ to obtain $c_1, \ldots, c_n$,
– It permutes them to $\grave{c_1}, \ldots, \grave{c_n}$.
– It queries the zero knowledge oracle for the disjunctive proofs of each cipher.
– Computes the Zero knowledge of the sum as done in Figure 2.
– Sends the proofs together with $\grave{c_1}, \ldots, \grave{c_n}$ to $\mathcal{A}$.
– If the proofs are real then $\mathcal{A}$ should return back the expected permutation, otherwise $\mathcal{A}$ gives a guess which is unlikely to be the permutation (probability is $n!$) and implies the zero knowledge proofs were a simulation only.

B Non-interactive Zero knowledge proofs

Equality between discrete logs: Proving knowledge of the discrete logarithm x to bases $f, g \in \mathbb{Z}_p^*$, given h, k where $h \equiv f^x \bmod p$ and $k \equiv g^x \bmod p$ [15, 6].

Sign. Given f, g, x, select a random nonce $w \in_R \mathbb{Z}_q^*$. Compute Witnesses $f' = f^w \bmod p$ and $g' = g^w \bmod p$, Challenge $c = \mathcal{H}(f', g') \bmod q$ and Response $s = w + c \cdot x \bmod q$. Output signature as (f', g', s)
Verify. Given f, g, h, k and signature (f', g', s, c), check $f^s \equiv f' \cdot h^c \pmod p$ and $g^s \equiv g' \cdot k^c \pmod p$, where $c = \mathcal{H}(f', g') \bmod q$.

A valid proof asserts $\log_f h = \log_g k$; that is, there exists x, such that $h \equiv f^x \bmod p$ and $k \equiv g^x \bmod p$.
Re-Encryption proofs for Exponential ElGamal Imagine you have two Exponential ElGamal ciphertexts for the key-pairs$(y = g^x, x)$:

1. $c = (u_1, v_1) = (g^{r_1}, y^{r_1} g^m)$,
2. $\grave{c} = (u_2, v_2) = (g^{r_1 + r_2}, y^{r_1 + r_2} g^m)$.

In other words $\grave{c} = ReEnc(c, r_2)$. A zero knowledge proof of Re-Encryption is simply done by the prover providing a zero knowledge proof of the equality between discrete logs between u_2/u_1 and v_2/v_1 to the bases g, y respectively.
Disjunctive Re-encryption Proof for Exponential ElGamal Let $h = g^y$. Given $(u_i, v_i) = (g^x g^{\zeta}, h^x \cdot h^{\zeta} \cdot g^m)$ is a re-encryption of $(u, v) = (g^x, h^x \cdot g^m)$ for a random $\zeta \in \mathbb{Z}_p^*$. Prove that (u_i, v_i) belongs to the list $\{(u_1, v_1), \ldots, (u_n, v_n)\}$ [14].

Sign. Select random values $d_1, \ldots, d_n$, $r_1, \ldots, r_n \in \mathbb{Z}_p^*$. Compute $a_t = (\frac{u_t}{u})^{d_t} g^{r_t}$, $b_t = (\frac{v_t}{v})^{d_t} h^{r_t}$ where $t \in \{1, \ldots, i-1, i+1, \ldots, n\}$. Choose randomly a nounce $\omega \in \mathbb{Z}_p^*$. Let $a_i = g^{\omega}$ and $b_i = h^{\omega}$. Compute challenge $c = \mathcal{H}(E||a_1|| \ldots ||a_n||b_1|| \ldots ||b_n)$ where $E = (u||v||u_1||v_1|| \ldots ||u_n||v_n)$. Compute $d_i = c - \sum_{t=1, t \neq i}^{n} d_y$. Compute $r_i = \omega - \zeta d_i$ then Witnesses $d_1, \ldots, d_n$, Challenge c and Response $r_1, \ldots, r_n$. Output signature of knowledge (r_t, d_t) where $t \in [1, n]$

Verify. Let $E1 = (\frac{u_1}{u})^{d_1} g^{r_1} || \ldots ||(\frac{u_n}{u})^{d_n} g^{r_n}$. Let $E2 = (\frac{v_1}{v})^{d_1} g^{r_1} || \ldots ||(\frac{v_n}{v})^{d_n} g^{r_n}$. Check $\sum_{t=1}^{n} d_t = \mathcal{H}(E||E1||E2)$

A valid proof asserts that $(u_i, v_i) \in \{(u_1, v_1), \ldots, (u_n, v_n)\}$.

Disjunctive Double Re-Encryption Proofs for Exponential ElGamal
Given the following:

- Let $h = g^y$. Let $(CT, \bar{CT})$ be a pair of ElGamal Encryption for the same message m.
- Let $CT_i = (u_i, v_i) = (g^{x_1} g^{\zeta_1}, h^{x_1} \cdot h^{\zeta_1} \cdot g^m)$ be a re-encryption of $(u, v) = (g^{x_1}, h^{x_1} \cdot g^m)$ for a random $\zeta_1 \in \mathbb{Z}_p^*$.
- Let $\bar{CT_i} = (\bar{u_i}, \bar{v_i}) = (g^{x_2} g^{\zeta_2}, h^{x_2} \cdot h^{\zeta_2} \cdot g^m)$ be a re-encryption of $(\bar{u}, \bar{v}) = (g^{x_2}, h^{x_2} \cdot g^m)$ for a random $\zeta_2 \in \mathbb{Z}_p^*$.

Prove that $(CT_i, \bar{CT_i})$ belongs to the list $\{(CT_1, \bar{CT_1}), \ldots, (CT_n, \bar{CT_n})\}$.

Sign. Select random values $d_1, \ldots, d_n$, $r_1, \ldots, r_n$, $R_1, \ldots, R_n \in \mathbb{Z}_p^*$.

For $t \in \{1, \ldots, i-1, i+1, \ldots, n\}$, compute:

$\alpha_t = (\frac{u_t \cdot v_t}{u \cdot v})^{d_t} (g \cdot h)^{r_t}$ and $\beta_t = (\frac{\bar{u_t} \cdot \bar{v_t}}{\bar{u} \cdot \bar{v}})^{d_t} (g \cdot h)^{R_t}$ and

$\delta_t = (\frac{u_t \cdot v_t \cdot v}{\bar{u_t} \cdot \bar{v_t} \cdot u})^{d_t} (g \cdot h)^{R_t - r_t}$

Choose randomly the nounces $\omega_1, \omega_2 \in \mathbb{Z}_p^*$. Let $\alpha_i = (g \cdot h)^{\omega_1}$ and $\beta_i = (g \cdot h)^{\omega_2}$, and $\delta_i = \frac{\beta_i}{\alpha_i}$.

Compute challenge $c = \mathcal{H}(E||\alpha_1|| \ldots ||\alpha_n||\beta_1|| \ldots ||\beta_n||\delta_1|| \ldots ||\delta_n)$ where $E = (u||v||u_1||v_1|| \ldots ||u_n||v_n)$. Let $d_i = c - \sum_{t=1, t \neq i}^{n} d_y$, $r_i = \omega_1 - \zeta_1 d_i$, and $R_i = \omega_2 - \zeta_2 d_i$. Witnesses is $d_1, \ldots, d_n$, Challenge is c and Response is $r_1, \ldots, r_n$, $R_1, \ldots, R_n$.

Output signature of knowledge (r_t, d_t, R_t) where $t \in [1, n]$

Verify. Let $E_1 = ((\frac{u_1 \cdot v_1}{u \cdot v})^{d_1} (g \cdot h)^{r_1} || \ldots ||(\frac{u_n \cdot v_n}{u \cdot v})^{d_n} (g \cdot h)^{r_n})$. Let $E_2 = ((\frac{\bar{u_1} \cdot \bar{v_1}}{\bar{u} \cdot \bar{v}})^{d_1} (g \cdot h)^{R_1} || \ldots ||(\frac{\bar{u_n} \cdot \bar{v_n}}{\bar{u} \cdot \bar{v}})^{d_n} (g \cdot h)^{R_n})$. Let $E_3 = ((\frac{u_1 \cdot v_1}{\bar{u_1} \cdot \bar{v_1}} \frac{v}{u})^{d_1} (g \cdot h)^{R_1 - r_1} || \ldots ||(\frac{u_n \cdot v_n}{\bar{u_n} \cdot \bar{v_n}} \frac{v}{u})^{d_n} (g \cdot h)^{R_n - r_n})$. Check $\sum_{t=1}^{n} d_t = \mathcal{H}(E||E_1||E_2||E_3)$.

If true accept the signature of knowledge otherwise reject it.

A valid proof asserts that $(u_i, v_i) \in \{(u_1, v_1), \ldots, (u_n, v_n)\}$.