# Two-Party Generation of Shared RSA Keys through Encryption Switching Protocols

by
Nick Cunningham

A Thesis submitted in partial fulfillment of the requirements for Honors
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2017

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

There are cryptographic uses for RSA moduli such that no party knows the factorization. One such use is threshold cryptography [1]. Given an RSA instance with modulus $N$ of unknown factorization and a set of integers $d_i$ for $1 \leq i \leq k$ such that $\sum d_i = d \mod \phi(N)$, where $d$ is the decryption exponent, give each of the $k$ parties a unique $d_i$. It is now only possible to decrypt messages encrypted for the RSA instance, or to sign messages under the RSA instance's keys, through a $k$-party computation with the cooperation of all the parties with shares. This is suitable for any cryptographic keys that are sensitive and should be difficult for any one party to use.

Boneh and Franklin use Certificate Authority (CA) keys as an example in [1]. The keys of a CA are used for important authentication procedures on the internet; website public keys must be signed by a CA to be trusted by a regular browser. But this also means that a stolen or misused CA key is sufficient to allow man-in-the-middle attacks on most websites. Having such keys be shared between parties would mean that multiple different attacks must succeed in order to steal the keys, and additionally that no single person would be able to use their access to the key to impersonate a website.

## 1.2 Encryption Switching Protocols

In their paper, Couteau et al. [2] defined what they called *encryption switching protocols* (ESPs). Given two encryption schemes $\Pi_1, \Pi_2$, a *twin ciphertext pair* is a pair of two encryptions, one in each $\Pi_i$, of the same value. They define an encryption switching protocol between $\Pi_1, \Pi_2$ as a tuple $(\mathsf{Share}, \mathsf{Switch})$ such that:

$\mathsf{Share}$ takes the keys of the encryption schemes and outputs a secret sharing between Alice and Bob

Switch is a two-party protocol through which Alice and Bob may jointly compute from an encryption in $\Pi_1$ a twin ciphertext in $\Pi_2$ or vice versa.

They also define *computational equality* to be a property on two sets such that they are computationally equal if the probability that an adversary can output an element in their symmetric difference is negligible.

They then defined and proved the security of an encryption switching protocol between the Paillier encryption scheme encrypting messages in $\mathbb{Z}_n$ ($\mathcal{E}_\oplus$) and a modified version of the ElGamal encryption scheme with message space $\mathbb{Z}_n^* \cup \{0\}$ ($\mathcal{E}_\otimes$) for an RSA modulus $n$. To allow their modified ElGamal scheme to encrypt zero securely, they also define what they call an Encrypted Zero Test, which is a two-party computation that reveals no additional information but whether the value encrypted is equal to zero. They proved that this protocol is secure under the DDH (Decisional Diffie-Hellman) assumption in $\mathsf{QR}_n$, the QR (Quadratic Residuosity) assumption in $\mathbb{Z}_n^*$, the DCR (Decisional Composite Residuosity) assumption over $\mathbb{Z}_n^*$, and the DCR assumption over $\mathbb{Z}_N^*$ (with $N$ an additional RSA modulus used by the protocol).

They showed that $\mathbb{Z}_n$ and $\mathbb{Z}_n^* \cup \{0\}$ are computationally equal, because elements in the symmetric difference would allow factorization of $n$.

These two encryption schemes have the properties of additive and multiplicative homomorphism, respectively; that is, they have defined operations $\boxplus, \boxtimes$ such that for messages $m_i$, randomness $r_i$, and some operation $\odot$

$$\mathcal{E}_\oplus(m_1; r_1) \boxplus \mathcal{E}_\oplus(m_2; r_2) = \mathcal{E}_\oplus(m_1 + m_2; r_1 \odot r_2)$$

$$\mathcal{E}_\otimes(m_3; r_3) \boxtimes \mathcal{E}_\otimes(m_4; r_4) = \mathcal{E}_\otimes(m_3 + m_4; r_3 \odot r_4)$$

Additionally, they can be randomized; that is, there is a function $\mathsf{Rand}(\mathcal{E}(m; r), r')$ which, given an encryption of $m$ and new randomness $r'$, uses the identity homomorphism to produce a new encryption of $m$ statistically indistinguishable from encrypting $m$ with new randomness $r'$.

Also, since both encryption schemes are two-party, they are such that either Alice or Bob may encrypt values, but neither may decrypt without performing a decryption protocol that gives the result to one or both.

This allows 2-party computation to be done by taking advantage of the homomorphisms. Couteau et al. [2] also show how to use the homomorphisms to perform zero-knowledge proofs, most importantly what they call a *twin-ciphertext proof*, which allows a prover to prove that two encryptions, one in each encryption scheme, are a twin-ciphertext pair.

## 1.3 Efficient Generation of Shared RSA Keys

Boneh and Franklin [1] created a protocol that allows the generation of a shared RSA key. Their method requires 3 parties (Alice, Bob, and a helper Henry), and gives shares of the key to Alice and Bob (but not Henry). In their protocol, there are a series of private distributed computations done:

1. Alice and Bob pick random integer shares that they keep secret, with Alice's shares being equal to 3 mod 4 and Bob's being equal to 0 mod 4 (this means that the modulus, if properly formed, will be a Blum integer.) With the help of Henry, they perform a computation to ensure the sum of the shares is not divisible by a small prime (for efficiency purposes). They do this process twice, producing $p_a, q_a$ for Alice and $p_b, q_b$ for Bob.

2. Alice, Bob, and Henry compute the RSA modulus $N = (p_a + p_b) \cdot (q_a + q_b)$ without revealing additional information about the shares.

3. Alice and Bob together test whether $N$ is the product of two primes. If not, they restart the protocol.

4. Alice, Bob, and Henry calculate the public encryption exponent $e$ and shares for Alice and Bob of the decryption exponent $d$.

Their protocol is secure so long as at least two of the participants are not coordinating against the third.

## 1.4 Our Contribution

We can improve upon Boneh and Franklin's protocol for RSA key generation by using an encryption switching protocol. With access to homomorphisms, we can perform similar computations without requiring an online helper Henry, instead only needing Alice and Bob. We will assume that the ESP has been previously set up and can encrypt a much larger space than RSA modulus calculated by the protocol. This is reasonable because we want to be able to use the ESP to generate many other shared keys, so it is fair to require its modulus to be larger (that is, for it to have a longer keysize and be harder to crack). This does require an honest dealer for the setup phase described by Couteau et al. of the ESP, but after that has been done once previously, Alice and Bob will be able to generate shared RSA keys whenever they want to without a helper by using the ESP.

First we will discuss the case in which both parties are honest-but-curious, and discuss the situation in which they are malicious afterwards. Going section-by-section, we will provide alternate computations using the ESP rather than a helper or third party to perform the same actions from Boneh and Franklin's protocol.

# Chapter 2

# Distr. RSA Key Generation for Semi-Honest Participants

Recall that Boneh and Franklin's protocol consists of four main steps in order: trial division, calculation of modulus, modulus testing, and exponent calculation. We will show how to eliminate Henry from the steps in which he appears in the original protocol.

## 2.1 Trial Division

If all we attempt to do is generate two random shared integers and check whether their product is the product of two primes, it will take a large number of attempts to find a good modulus. A much more efficient way to generate a good modulus is first testing each shared integer to ensure that they are not products of small primes. This will save a large number of attempts from slow failure (for Boneh and Franklin, over a thousand-times speedup [1]).

So, we want to stop the protocol early if $q$ divisible by a small prime, with $j$ primes to be considered. In the original protocol, for each of $j$ primes:

1) Alice picks random $c_i \in Z_{p_i}$, $d_i \in Z_{p_i}^*$. For all $i \in [1, j]$, she computes $u_i = c_i + d_i q_a \mod p_i$, and sends $c_i, d_i$ to Bob and $u_i$ to Henry.

2) Bob computes $v_i = c_i - d_i q_b \mod p_i$ for all $i \in [1, j]$. He sends $v_i$ to Henry.

3) Henry declares success if for all $i$, $u_i \neq v_i$, and otherwise declares failure.

This protocol is correct because for the prime $p_i$, it verifies that $q_a \neq -q_b \mod p_i$; that is, $q_a + q_b \neq 0 \mod p_i$. So, if Henry declares success, $q = q_a + q_b$ is not divisible by any primes tested.

We will modify the protocol to not require Henry:

1. Alice picks random $c_i \in Z_{p_i}$, $d_i \in Z_{p_i}^*$. For all $i \in [1, j]$, she computes $u_i = \mathcal{E}_{\oplus}(c_i + d_i q_a \mod p_i)$, and sends $c_i, d_i, u_i, r_{ai}$ to Bob.

2. Bob computes $v_i = \mathcal{E}_{\oplus}(-(-c_i + d_i q_b \mod p_i))$ for all $i \in [1, j]$. He sends $v_i$ to Alice.

3. Alice and Bob calculate $w_i = u_i \boxplus v_i$. They then perform an encrypted zero test (EZT) as defined in [2] and decrypt the result. If the result is 0, the test fails; otherwise, they continue.

This modified protocol is correct under the same logic as before. Also, WLOG Alice can simulate her view of the protocol given the final result that would be revealed:

If the protocol would pass at the $i$th small prime $p_i$, the simulator sets $v_i$ to be an encryption of a small positive value; and since $p_i$ is much smaller than the size of the ESP message space, and Alice is honest-but-curious, the sum $u_i \boxplus v_i$ must encrypt a small positive value (relative to the message space) instead of zero.

If the protocol would fail at prime $p_i$, then instead of calculating $v_i$ correctly, since Alice is honest-but-curious, the simulator uses Alice's share to calculate the value encrypted by $u_i$ and sets $v_i$ such that $u_i \boxplus v_i = \mathcal{E}_{\oplus}(0)$.

Since the ESP cryptosystems are IND-CPA secure [2], being able to distinguish $v_i$ generated in this way from $v_i$ generated correctly would break our security properties.

Symmetrically, Bob's transcript can be simulated; the simulator would follow the protocol honestly except that it would manipulate encryptions to be of a very small negative number to force non-zero success, and change $u_i$ to be such that $u_i \boxplus v_i = \mathcal{E}_{\oplus}(0)$ to force failure. Since Bob is honest-but-curious, it can calculate the necessary values of $u_i$ to ensure the correct result.

Finally, we know that the EZT can be simulated [2].

So, given the knowledge of whether the test fails (and at what prime), each party can simulate their transcript of the $u_i$ and $v_i$ exchange such that the simulated sum is zero if and only if the sum done properly is zero. Then, the EZT is similarly simulatable, so the entire protocol is simulatable.

## 2.2 Distributed Computation of the Modulus

In the original protocol, Boneh and Franklin use a protocol involving the evaluation of polynomials to calculate the modulus. Our solution is very dissimilar, though; thanks to the homomorphic properties we have, it is very simple to calculate $N$.

So, while computing N, we will eliminate the helper. Alice and Bob possess shares $p_a, q_a$ or $p_b, q_b$ such that $p_a + p_b = p$ and $q_a + q_b = q$ where $N = pq$. This can be done simply with an encryption switching protocol that has already been set up, so long as its own modulus is large enough. This procedure works as follows:

1. Alice encrypts $p_a$ in the additively homomorphic scheme. She sends $E_{\oplus}(p_a)$ to Bob.

2. Bob encrypts $p_b$ in the additive scheme and sends it to Alice. They both compute

$$E_{\oplus}(p) = E_{\oplus}(p_a) \boxplus E_{\oplus}(p_b) = E_{\oplus}(p_a + p_b)$$

3. The procedure is repeated for the shares of $q$.

4. Alice and Bob use the ESP to switch $E_{\oplus}(p)$ and $E_{\oplus}(q)$ into $E_{\otimes}(p)$ and $E_{\otimes}(q)$.

5. They then calculate
$$E_{\otimes}(pq = N) = E_{\otimes}(p) \boxtimes E_{\otimes}(q)$$

6. Finally, they jointly decrypt $E_{\otimes}(N)$ to both of them to get the RSA modulus.

Couteau et al. proved that functions evaluated on ciphertexts using homomorphic addition or multiplication and switching in either direction plus a final decryption are input-indistinguishable [2]. This is because both decryption and switching can be simulated such that they are indistinguishable from a correct decryption and switching, except that they force the output to be a particular plaintext or ciphertext. So, no information is gained from this protocol by Alice or Bob about the other's inputs except that they are of the form $(p_a + p_b)(q_a + q_b) = N$.

Suppose WLOG that Alice, given random $q_a, p_a$ and $N = pq$ such that $p = p_a + p_b$, $q = q_a + q_b$ are prime, can compute $q_b$ or $p_b$ with non-negligible probability. Then, she can compute $p_b = N/(q_a + q_b) - p_a$ or similarly $q_b$, and thus have both $p, q$. But this is equivalent to factoring, since given a modulus $N$ she could simply sample random $q_a, p_a$ and use her algorithm to factor $N$; and it is assumed that there is no probabilistic polynomial time algorithm that can factor $N$ with non-negligible probability, so Alice must not be able to compute $q_b$ or $p_b$ with non-negligible probability.

## 2.3 Distributed Primality Test

The distributed primality test can be done as Boneh and Franklin do [1].

1. Once both parties know N, the two pick a random $g \in \mathbb{Z}_N^*$.

2. Alice computes the Jacobi symbol of $\left(\frac{g}{N}\right)$. If it is not 1, restart with a new $g$.

3. Alice computes $v_a = g^{(N-p_a-q_a+1)/4} \mod N$. Bob computes $v_b = g^{(p_b+q_b)/4} \mod N$.

4. Alice and Bob send each other $v_a$ or $v_b$ respectively. They test whether $v_a = \pm v_b \mod N$; if not, they say $N$ is not a product of two primes, and if so, they say the test suceeded.

Boneh and Franklin go on to prove that the test always succeeds if $N$ is a product of two primes generated from the earlier random picking of shares, and that the test does not fail with exponentially small probability if $N$ is not. Additionally, they show that if $p, q$ are prime, both parties can simulate their transcript of the protocol by computing (WLOG) $v_a$ as normal, then randomly setting $v_b = \pm v_a$ (equivalent to guessing whether $g$ is a quadratic residue modulo $N$); so, this preserves privacy.

## 2.4 Calculating shares of private exponent

Alice and Bob now need to calculate a shared decryption exponent $d = d_a + d_b$. For small public exponents, Boneh and Franklin have a protocol which can securely calculate the private exponent with only the participation of Alice and Bob; so, for such cases, we can use it directly. However, for arbitrary public exponents, they require Henry as well. We will look at this computation and modify it.

Suppose they have already agreed upon a public encryption exponent $e$. Let $\phi_a = (N - p_A - q_a + 1)$ and $\phi_b = (-p_b - q_b)$. Their sum is equal to $\phi(N)$. In the original, the protocol for arbitrary public exponent is as follows [1]:

1. Alice picks random $r_a \in \mathbb{Z}_e$. Bob picks random $r_b \in \mathbb{Z}_e$.
2. Use the protocol that was used to calculate $N$ to calculate $\Psi = (r_a + r_b)(\phi_a + \phi_b) \mod e$. This requires the help of Henry. Give the result to Alice and Bob. If it is not invertible modulo $e$, restart.
3. Alice calculates $\zeta_a = r_a \Psi^{-1} \mod e$, and Bob $\zeta_b = r_b \Psi^{-1} \mod e$. This means
$$\zeta_a + \zeta_b = (r_a + r_b)\Psi^{-1} = \phi^{-1} \mod e$$
4. They pick a large prime $P > 2N^2 e$ and again use a slightly modified version of their protocol for calculation of modulus (requiring Henry's help) to create a sharing:
$$A + B = -(\zeta_a + \zeta_b)(\phi_a + \phi_b) + 1 \mod P$$
where Alice knows A and Bob knows B.
5. With probability greater than $1 - \frac{1}{N}$, $A + B > P$, so Alice sets $A \leftarrow A - P$. If this does not work, a trial decryption will discover the fact and the calculation may be tried again.
6. $e$ divides $A + B$ because
$$A + B = -(\zeta_a + \zeta_b)(\phi_a + \phi_b) + 1 = -(\phi_a + \phi_b)^{-1}(\phi_a + \phi_b) + 1 = 0 \mod e$$
So, Alice and Bob set $d_a = \lfloor A/e \rfloor$ and $d_b = \lceil B/e \rceil$, respectively. This works because $de = A + B = k\phi(N) + 1 = 1 \mod \phi(N)$. So, the decryption shares are such that $d = d_a + d_b$.

We must change this protocol in a few areas. Unfortunately, we must reduce the ceiling of exponent size from "arbitrary" to "very large" to have a functioning version. Our protocol will do the following:

1. Let $S$ be the size of the message space of our ESP. Let $s = \lfloor \frac{S - 2Ne}{2e} \rfloor$. This means that $2se + 2Ne$ is a value in our ESP message space; and because $\phi(N) < N$, this means that $2se + 2\phi(N)e$ also is in the ESP message space. We have previously assumed that the ESP encrypts a much larger space than the modulus. We will suppose that it is large enough to guarantee $s > (Ne)^2$; that is, $s$ is at least twice as long as the modulus times the encryption exponent. Alice picks random $r_a \in \mathbb{Z}_e$, $r'_a \in \mathbb{Z}_s$. Bob picks random $r_b \in \mathbb{Z}_e$, $r'_b \in \mathbb{Z}_s$.

2. Use the homomorphisms in a similar way as was used to calculate $N$ to calculate $\Psi' = (r'_a + r'_b)e + (r_a + r_b)(\phi_a + \phi_b)$. This is at most $2se + 2\phi(N)e$, and thus is a value over the integers that can be encrypted in our ESP space without modular reduction. Give the result to Alice and Bob. They calculate $\Psi = \Psi' \mod e$. If it is not invertible modulo $e$, restart.

3. Alice calculates $\zeta_a = r_a\Psi^{-1} \mod e$, and Bob $\zeta_b = r_b\Psi^{-1} \mod e$.

4. Again, the ESP encrypts a much larger space than the modulus. So, it is safe for us to calculate in our ESP
$$\mathcal{E}_\oplus(-(\zeta_a + \zeta_b)(\phi_a + \phi_b) + 1)$$
similarly to how we calculated the modulus. Alice picks a random integer $A$ in the space of the ESP. She then encrypts $\mathcal{E}_\oplus(-A)$ and sends it to Bob. They use the homomorphism to calculate
$$\mathcal{E}_\oplus(-(\zeta_a + \zeta_b)(\phi_a + \phi_b) + 1) \boxplus \mathcal{E}_\oplus(-A) = \mathcal{E}_\oplus(B)$$
and give the decryption of $\mathcal{E}_\oplus(B) = B$ to Bob (but not Alice). It must therefore be the case that
$$A + B = -(\zeta_a + \zeta_b)(\phi_a + \phi_b) + 1$$

5. $e$ divides $A+B$ for the same reasons as in the original protocol. So, Alice and Bob set $d_a = \lfloor A/e \rfloor$ and $d_b = \lceil B/e \rceil$, respectively. The decryption shares are again such that $d = d_a + d_b$.

This protocol is correct for the same reasons as the original protocol.

As before, since functions evaluated through the homomorphisms are input-indistinguishable, the only useful information gained are the values $\Psi'$ and $A$ or $B$, respectively. $\Psi'$, though, is randomized: since $r_a, r_b$ are sampled uniformly from $\mathbb{Z}$, their sum appears uniformly distributed mod $e$ to both Alice and Bob.

Additionally, $r'_a, r'_b$ are sampled from $\mathbb{Z}_s$. Suppose that we know $\Psi$, the remainder when we take $\Psi'$ modulo $e$. $\Psi' = (r'_a + r'_b)e + (r_a + r_b)\phi(N) = \Psi + (r'_a + r'_b + k)e$ for some integer $k$. We know that $\Psi = \Psi' \mod e$ already.

WLOG, this means Alice can calculate $\phi(N)r_b \mod e$. However, as in the original, this is safe to leak because $\gcd(N, e) = 1$ and $\Psi$ is invertible modulo $e$, so $\phi(N) \mod e$ could be any element of $Z_e^*$; so, $\phi(N)r_b \mod e$ and a random element of $Z_e^*$ are indistinguishable and $\Psi$ does not leak information

So, to recover information about $\phi(N)$, we must recover information about $k$, the additional information from $\Psi'$. We know $(r'_a + r'_b + k)$; so, WLOG, suppose we are Alice. We can subtract $r'_a$ to get $(r'_b + k)$

Now, let $Dist$ be the distribution of $(r'_b + k)$ and $Dist'$ be the distribution of $r'_b + r_c$, for $r_c$ sampled uniformly at random from $\mathbb{Z}_{2N}$ (up to the maximum value of $(r_a + r_b)\phi(N)$. Since $r'_b$ was sampled from $\mathbb{Z}_s$, and $s > (Ne)^2$, these distributions are identical for all possible values on the range $[4N, s - 2N]$, while potentially taking values on the range $[0, s + 2N]$. So, they can only be distinguished through the values at the edge.

However, $Dist'$ and $Dist$ only take those values with probability less than $\frac{s-4N}{s} = 1 - \frac{4N}{s}$ (the probability that $s$ does not force the result to be in the safe range). But this is at most $1 - \frac{4N}{(Ne)^2} = 1 - \frac{4}{Ne^2}$, which is negligibly likely.

So, there is a negligible chance that the distributions could differ in revealing ways; so they are statistically indistinguishable. This means that no useful information is revealed by exposing $\Psi'$ in addition to $\Psi$, because the extra information is statistically indistinguishable from a simulatable random distribution.

Now, suppose given $A$ we could calculate $B$ or equivalently $d_b$. $A$ does not depend on $B$; so this would mean that given a random number drawn uniformly from some large space, a public exponent $e$ and an RSA modulus $N$, we could calculate the decryption exponent $d = (A + B)/e$. But this would break RSA, which is assumed to be secure.

Similarly, if $A$ is drawn uniformly at random, $B$ will also be distributed uniformly at random over the ESP message space because in general, with uniformly drawn $r \in Z_m$ and constant $k \in Z_m$, the distribution of $r + k \mod m$ is uniform as well. So, the same argument holds for Bob's knowledge being insufficient to calculate $A$ or $d_a$.

Also, after $\Psi$ is calculated, Alice sees no new results of decryption, only applying encryptions, switches, and homomorphisms; so, her transcript afterwards is simulatable.

Bob sees the result of a decryption to $B$, his share; but this is also simulatable, because the simulation can force decryption to $B$ [2].

Combining all of these, we know:

1. Operations done with the ciphertexts are input-indistinguishable.

2. $\Psi$ is statistically indistinguishable from a uniform sampling of $Z_e^*$.

3. For Alice, $\Psi' - \Psi - r'_a e$ is statistically indistinguishable from $r'_b + r_c$ for $r'_b \leftarrow \mathbb{Z}_s$ and $r_c \leftarrow \mathbb{Z}_{2N}$ uniformly at random. Similarly, exchanging Bob's shares for Alice's provide the same indistinguishability to Bob.

4. Under the RSA assumption, neither Alice nor Bob can compute $B$ or $A$ from $A$ or $B$ and public knowledge, respectively; and in fact they can simulate the part of the transcript that results in them getting their share.

So, we can simulate this protocol by forcing $\Psi$ and $\Psi'$ to be random samples from the respective distributions from which they are statistically indistinguishable when we decrypt them, having the actual input ciphertexts of the other party be random, and (for Bob) forcing the final decryption to produce $B$. The actual application of the homomorphisms can be done as normal.

So, this protocol is secure and leaks no useful information.

# Chapter 3

# Malicious Adversaries

## 3.1 Zero Knowledge Proof of Knowledge of the Committed Value

Couteau et al. [2] gave ZK proofs using an additively homomorphic commitment scheme, allowing the use of $\mathcal{E}_{\oplus}$. For the purpose of avoiding malicious behavior, we can use zero-knowledge proofs. We will use the following zero-knowledge proof to do so.

Let $n$ be the integer such that homomorphic computations and encryptions are done modulo $n$.

Alice has committed $C_a = \mathcal{E}_{\oplus}(a; \text{rand}_a)$ and wishes to prove that she knows its value. To do so, she generates a uniformly random number $r \leftarrow \mathbb{Z}_n$. She then calculates

$$C_r = \mathcal{E}_{\oplus}(r; \text{rand}_r)$$

and sends it to Bob.

Bob then generates $b \leftarrow \mathbb{Z}_n^*$ uniformly at random, and sends it to Alice.

Alice calculates:

$$s = ab + r$$

$$C_s = (C_a \bullet b) \boxplus C_r = \mathcal{E}_{\oplus}(ab + r; \text{rand}_{ab+r})$$

Since Alice knows the random coins for $C_a, C_r$, she can also calculate $\text{rand}_{ab+r}$. She sends Bob $s, \text{rand}_{ab+r}$.

Finally, Bob verifies that

$$(C_a \bullet b) \boxplus C_r = \mathcal{E}_{\oplus}(s; \text{rand}_{ab+r})$$

If it is, then he accepts the proof.

If Alice knows $a, \text{rand}_a$, then because of the homomorphisms, her output will always be such that Bob accepts the proof.

Suppose that Alice does not know $a, \text{rand}_a$. She will still be able to calculate $C_s$ correctly. However, extracting the randomness of a ciphertext is sufficient to determine whether that ciphertext

encrypts a particular plaintext, and therefore breaks the IND-CPA security of our encryption scheme. So, Alice cannot calculate $\text{rand}_{ab+r}$.

Similarly, $b, r$ are sampled independently from $a$. So, it must not be the case that knowing $C_a$, $C_s$ and the fact that $s = ab + r$ allows her to recover $a$. This is because otherwise taking two random numbers and applying the homomorphism would be sufficient to decrypt values, which obviously breaks the security of our encryption scheme. Since she is not able to recover $a$ and $a = \frac{s-r}{b}$, she must also not be able to recover $s$ or she could simply solve the equation.

So, if Alice does not know $a, \text{rand}_a$, she is unable to calculate $s, \text{rand}_{ab+r}$ and therefore cannot provide Bob with results that he would accept.

So Alice can convince Bob to accept the proof if and only if she knows the values $a, \text{rand}_a$.

Additionally, this is zero-knowledge. The simulator can create a transcript with the same distribution: First, it picks $s' \leftarrow \mathbb{Z}_n$ and $b \leftarrow \mathbb{Z}_n^*$ uniformly at random. It then calculates

$$C'_s = \mathcal{E}_{\oplus}(s', \text{rand}'_s)$$

$$C'_r = C'_s \boxminus (C_a \bullet b)$$

This is possible because the Paillier cryptosystem has a subtractive in addition to additive homomorphism. However, since $b$ is given to Alice in the original after she has committed to $r$, she cannot fake a proof this way. Then, the simulated transcript is just:

1. Alice sends Bob $C'_r$.
2. Bob sends Alice $b$.
3. Alice sends Bob $s', \text{rand}'_s$.

In the real proof, $s$ is distributed uniformly at random over $\mathbb{Z}_n$ since $r$ is as well, and $s = ab + r$ and computations are modulo $n$. So, it is distributed identically to $s'$, which is sampled from $\mathbb{Z}_n$ uniformly at random.

From our security properties, we know that $\text{rand}_{ab+r}$ and $\text{rand}'_s$ are statistically indistinguishable, because otherwise we would have a non-negligible advantage distinguishing between ciphertexts which have been freshly encrypted and those that are the result of homomorphisms with inputs that are not fully known.

Similarly, $C_r$ and $C'_r$ are encryptions under an IND-CPA scheme, and their distributions are therefore statistically indistinguishable.

Finally, $b$ is generated from the same distributions in both the real proof and the simulation.

So, all simulated values have indistinguishable distributions; and therefore Bob can correctly simulate the proof, and it is zero-knowledge.

In the case where the committed value $C_a$ is in the multiplicative scheme, we can have Alice create a new encryption of $C'_a$ in the additive scheme and perform the above proof; then, Alice and Bob will perform a twin-ciphertext proof as in [2] that will prove that $C_a$ and $C'_a$ encrypt the same value.

## 3.2 Security against Malicious Adversaries

In general, our modified protocol can be made secure against malicious adversaries by using zero-knowledge proofs. Whenever Alice or Bob encrypts a value that they want to use in a computation, they perform a ZK proof that they know the value encrypted; this prevents them from doing computations with values they do not already know (and thus from gaining knowledge about ciphertexts computed or sent by the other). This, combined with the fact that the homomorphism is deterministic, guarantees that all inputs to the calculations done are known by the party giving them as input (thus preventing reuse of unknown quantities from previous calculations) and that the operations are done correctly.

Additionally, for a calculation to reveal information about keys generated in previous runs of the protocol, it must necessarily involve information known only to the non-adversarial party; otherwise no information could be revealed. But because the values used are everywhere randomly generated such that both parties contribute entropy, there is no way to force the other party to use the same values as a previous run, and it is negligibly likely that a party will generate the same random value as previously.

So, the only feasible way to learn information about a previous run would be to use encrypted values from that run, but we prevent this through our ZK proofs and the deterministic homomorphism.

# Chapter 4

# Conclusion

Using ESPs make it much more simple and efficient to do secure two-party computations when the computations involve addition and multiplication modulo some large $N$. We have used them to remove the requirement of a helper third party for generation of shared RSA keys, but there are many other contexts in which they could be applied. Couteau et al., for instance, mention private disjointness testing, which allows two players to securely determine whether their respective sets share an element or not [2]. They also mention zero-knowledge proofs, and show that ESPs provide simpler ways to prove things like knowledge of exponential relationships of committed values and other mathematical properties involving addition and multiplication.

# Bibliography

[1] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. *J. ACM*, 48(4):702–722, July 2001.

[2] Geoffroy Couteau, Thomas Peters, and David Pointcheval. Encryption Switching Protocols. In Matthew Robshaw and Jonathan Katz, editors, *Crypto 2016 - 36th Annual International Cryptology Conference*, Santa Barbara, United States, August 2016. Springer.