Secure Two-party Dilithium Signing Protocol

Yu Fu, Xiufeng Zhao

The PLA SSF Information Engineering University
Zhengzhou, China
e-mail: zhao_xiu_feng@163.com

2021 17th International Conference on Computational Intelligence and Security (CIS) | 978-1-6654-9489-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/CIS54983.2021.00098

***Abstract*—The National Institute of Standards and Technology is in the process of selecting post-quantum public-key cryptography (PQC) algorithms through a public competition-like process. Dilithium, a lattice-based signature scheme, is a PQC candidate submitted to the NIST. It offers a fairly good performance with relatively straightforward implementation. In this study, we considered a secure two-party Dilithium signing protocol. We described how to build a distributed key generation and distributed signing protocol for Dilithium. Our protocol achieves a coordinated two-party Dilithium signature generation using Brakersiki/Fan-Vercauteren homomorphic encryption scheme. Our protocol is proven secure against active adversaries corrupting one of the players. Compared with other protocols, our protocol has advantage in resisting quantum computing attacks.**

Dilithium, distributed signature, homomorphic encryption.

I. INTRODUCTION

Digital signatures are applied for identity authentication, data integrity, non-repudiation, and anonymity. The user generates the signature of a message using a private key; the verifier confirms the signature's validity with the public key. Nevertheless, it is difficult to guarantee the storage security of the private key in mobile internet applications. Therefore, we split private key and generate signature coordinatively to response to advances in the development of mobile internet applications. For example, digital signatures are used in bitcoin, and the theft of a signing key immediately results in tangible financial loss. Bitcoin has a multisignature solution built-in, which is based on using multiple distinct signing keys. Nevertheless, threshold cryptography provides a more general solution.

Threshold secret sharing can be used to realize a joint digital signature in which the private key is divided into n private key shares and safely distributed to n participants. Furthermore, a threshold t is defined, which is the minimum number of n participants that can reconstruct the private key. However, after the private key is reconstructed, the party with the complete private key can independently sign the message without the knowledge of other participants, which is a huge security flaw.

Threshold cryptography protocols exist for various problems, such as RSA signing[1,2], Regev decryption [3]. Some works present two-party protocol for DSA/ECDSA signing[4-6]. However, above signature schemes cannot resist quantum attack. Dilithium is a PQC candidate submitted to the NIST, which is a lattice-based signature scheme constructed using the Fiat-Shamir heuristic, whose security is based on the hardness of the LWE problem[7]. Dilithium offers reasonably good performance and is relatively straightforward to implement. Therefore, it is necessary to construct a secure protocol for two-party Dilithium signing.

Our results. In this study, we consider a secure two-party Dilithium signing protocol. We describe how to build a distributed key generation protocol and distributed signing protocol for Dilithium. Our protocol uses the Brakerski/Fan-Vercauteren (BFV) homomorphic encryption scheme to achieve a coordinated two-party Dilithium signature generation. Based on the random oracle assumption, our protocol is proven secure against active adversaries corrupting one of the players. It delivers good performance in term of computation and communication costs. Furthermore, our protocol has advantage in resisting quantum computing attacks.

Organization. In section II, we give two-party Dilithium key generation and signing protocol. We give the security and performance analysis on the protocol in section III and section IV respectively. We give conclusions in section V.

II. TWO-PARTY DILITHIUM SIGNING PROTOCOL

In this section, we present our distributed two-party Dilithium signing protocol. The protocol includes two phases: the distributed key generation phase and the signing phase.

A. Distributed Key Generation

P_1 samples a random secret key vector $(\boldsymbol{s_{11}}, \boldsymbol{s_{12}})$ from $S_\eta^k \times S_\eta^l$ and computes $\boldsymbol{t_1} = \mathbf{A}\boldsymbol{s_{11}} + \boldsymbol{s_{12}}$, then commits $\boldsymbol{t}$ along with a zero-knowledge proof of knowledge $(\boldsymbol{s_{11}}, \boldsymbol{s_{12}})$ to P_2. P_2 samples a random secret key vector $(\boldsymbol{s_{21}}, \boldsymbol{s_{22}})$ from $S_\eta^k \times S_\eta^l$ and sends $\boldsymbol{t_2} = \mathbf{A}\boldsymbol{s_{21}} + \boldsymbol{s_{22}}$ along with a zero-knowledge proof of knowledge to P_1. Finally, P_1 decommits and generates BFV keys $(pk_{BFV}, evk_{BFV}, sk_{BFV})$ with a zero-knowledge proof of knowledge sk_{BFV} to P_2, and then P_2 verifies the proof. The output of the distributed key generation algorithm is

978-1-6654-9489-2/21/$31.00 ©2021 IEEE
DOI 10.1109/CIS54983.2021.00098

$vk = (\boldsymbol{A}, \boldsymbol{t} = \boldsymbol{t_1} + \boldsymbol{t_2})$, $sk_1 = (\boldsymbol{s_{11}}, \boldsymbol{s_{12}})$ and $sk_2 = (\boldsymbol{s_{21}}, \boldsymbol{s_{22}})$.

Distributed key generation can be completely simulated because of the extractability and equivocality of the proof and commitment. Assuming that P_1 is corrupted, receiving $\boldsymbol{t}$ from the trusted party, $\boldsymbol{t_1}$ and $(\boldsymbol{s_{11}}, \boldsymbol{s_{12}})$ from P_1, the simulator $\mathcal{S}$ defines the value sent by P_2 to be $\boldsymbol{t_2} = \boldsymbol{t} - \boldsymbol{t_1}$. Similarly, if P_2 is corrupted, the simulator commits to anything and then after seeing $(\boldsymbol{t_2}, \boldsymbol{s_{21}})$ as sent to the proof functionality, it defines $\boldsymbol{t_1} = \boldsymbol{t} - \boldsymbol{t_2}$.

B. Distributed signing

The signing protocol proceeds as follows. First, the parties run a similar "coin tossing protocol" as in the key generation phase to obtain a random $\boldsymbol{W} = \mathbf{A}\boldsymbol{y}$, which will be used in generating the distributed signature, and they store $\boldsymbol{y_1}$ and $\boldsymbol{y_2}$, respectively, where $\boldsymbol{y} \triangleq \boldsymbol{y_1} + \boldsymbol{y_2}$.

Second, P_1 computes $\boldsymbol{w_1} = HighBits(\boldsymbol{Ay})$ and a challenge $c = H(M||\boldsymbol{w_1})$, where M is the message to be signed. Then, P_1 computes $c_1 = Enc_{pk}^{BFV}(\boldsymbol{Ay_1} - \mathrm{c}\boldsymbol{s_{12}})$ and sends c_1 to P_2.

P_2 computes $\boldsymbol{w_1} = HighBits(\boldsymbol{Ay})$ and a challenge $c = H(M||\boldsymbol{w_1})$. Then, P_2 computes $\boldsymbol{z_2} = \boldsymbol{y_2} + c \cdot \boldsymbol{s_{21}}$ and $(\boldsymbol{r_1}, \boldsymbol{r_0}) \leftarrow Decompose_q(\boldsymbol{Ay_2} - c\boldsymbol{s_{22}}, 2\gamma_2)$, if $\|\boldsymbol{z_2}\|_\infty \geq \gamma_1 - \beta$ or $\|\boldsymbol{r_0}\|_\infty \geq \gamma_2 - \beta$ then return $\perp$. Otherwise, P_2 computes $c_2 = Enc_{pk}^{BFV}(\boldsymbol{z_2})$, $c_3 = c_1 \oplus Enc_{pk}^{BFV}(\boldsymbol{Ay_2} - \mathrm{c}\boldsymbol{s_{22}})$, $c_4 = HomEvel(evk_{BFV}, c_3, \|LowBits(\cdot, 2\gamma_2)\|_\infty)$, and $c_5 = HomEvel(evk_{BFV}, c_3, HighBits(\cdot, 2\gamma_2))$. Finally, P_2 sends c_2, c_4, c_5 to P_1.

P_1 can decrypt c_2, c_4, c_5 because he holds the BFV decryption key, $\boldsymbol{z_2'} = Dec_{sk}^{BFV}(c_2)$, $\|LowBits(\mathbf{A}\boldsymbol{y} - c \cdot (\boldsymbol{s_{12}} + \boldsymbol{s_{22}}), 2\gamma_2)\|_\infty = Dec_{sk}^{BFV}(c_4)$, and $HighBits(c \cdot (\boldsymbol{s_{12}} + \boldsymbol{s_{22}}), 2\gamma_2) = Dec_{sk}^{BFV}(c_5)$. Then, he computes $\boldsymbol{z} = (\boldsymbol{y_1} + c \cdot \boldsymbol{s_{11}}) + \boldsymbol{z_2'} = (\boldsymbol{y_1} + \boldsymbol{y_2}) + c \cdot (\boldsymbol{s_{11}} + \boldsymbol{s_{21}})$. Because $\boldsymbol{y} = \boldsymbol{y_1} + \boldsymbol{y_2}$, $\boldsymbol{z}$ is a valid Dilithium signature if it satisfies the following three conditions:

(1) $\|\boldsymbol{z}\|_\infty < \gamma_1 - \beta$,
(2) $\|LowBits(\boldsymbol{W} - c \cdot (\boldsymbol{s_{12}} + \boldsymbol{s_{22}}), 2\gamma_2)\|_\infty < \gamma_2 - \beta$,
(3) $HighBits(\boldsymbol{W} - c \cdot (\boldsymbol{s_{12}} + \boldsymbol{s_{22}}), 2\gamma_2) = HighBits(\boldsymbol{Ay})$.

However, P_2 may send an incorrect $\boldsymbol{z_2'}$ value to P_1 because P_1 can verify that the final signature is valid before outputting it; thus, a corrupt P_2 cannot cause P_1 to output invalid signature.

III. SECURITY ANALYSIS

In this section, we prove that Π (key generation and signing) is a secure two-party Dilithium signing protocol. The widely accepted ProVerif tool is an automatic cryptographic protocol verifier in the formal model. This protocol verifier is based on a representation of the protocol by Horn clauses. It can handle many different cryptographic primitives, including encryption, signatures, hash functions, and Diffie-Hellman key agreements, specified both as rewrite rules or as equations[17]. For example, the security analysis using ProVerif demonstrates that password-authenticated key exchange protocol has reached a proper level of efficiency without sacrificing the desired security properties. Otherwise, our two-party Dilithium signing protocol based on many fundamental protocol, such as commitment, zero-knowledge proof, and committed non-interactive knowledge proof. And it is difficult to specified as rewrite rules or as equations. Therefore, we demonstrate the security of our protocol using provable security theory similar to the work [5]. We conclude the protocol security analysis as a theorem.

Theorem 4.1. Assume that the BFV homomorphic encryption scheme is indistinguishable under chosen-plaintext attacks. Furthermore, assume that Dilithium is existentially-unforgeable under a chosen message attack, and that the zero-knowledge proofs and commitments are as described. Then, in the random oracle model, protocols 3.1 and 3.2 constitute a secure two-party protocol for distributed Dilithium signature generation.

Proof: We will prove that, for any given adversary $\mathcal{A}$ attacking the protocol, we construct a simulator $\mathcal{S}$ who forges a Dilithium signature in Experiment 4.1. As shown in Figure 4.1, with the probability that is negligibly close to the probability that $\mathcal{A}$ forges a signature in Experiment 4.2. Formally, we prove that if BFV is indistinguishable under chosen-plaintext attacks, for every PPT algorithm $\mathcal{A}$ and every $b \in \{1,2\}$, there exists a PPT algorithm $\mathcal{S}$ and a negligible function ε such that for every λ,

$$\left| \begin{matrix} Pr[Expt - Sign_{\mathcal{S},\pi}(1^\lambda) = 1] \\ -Pr[Expt - DistSign_{\mathcal{A},\Pi}^{b}(1^\lambda) = 1] \end{matrix} \right| \leq \varepsilon(\lambda) \quad (1)$$

where Π denotes Protocols 3.1 and 3.2 and π denotes the standard Dilithium signature scheme. By the assumption in the theorem, the Dilithium is secure, we have that there exists a negligible function ε' such that for every λ,

$$Pr[Expt - Sign_{\mathcal{S},\pi}(1^\lambda) = 1] \leq \varepsilon'(\lambda) \quad (2)$$

Combining this with Eq. (1), we conclude that

$$Pr[Expt - DistSign_{\mathcal{A},\Pi}^{b}(1^\lambda) = 1] \leq \varepsilon'(\lambda) + \varepsilon(\lambda) \quad (3)$$

According to Definition 2.2, Π is a secure two-party Dilithium signing protocol. We will prove Eq. (1) holds for $b = 1$ and $b = 2$, respectively.

Case 1: P_1 corrupted, $b = 1$. Let $\mathcal{A}$ be a PPT adversary in $Expt - DistSign_{\mathcal{A},\Pi}^{b=1}$. We will construct a PPT simulator $\mathcal{S}$ for $Expt - Sign_{\mathcal{S},\pi}$. The simulator $\mathcal{S}$ simulates the execution for $\mathcal{A}$, as described in the security of the protocol.

(1) In $Expt - Sign_{\mathcal{S},\pi}$, simulator $\mathcal{S}$ receives $(1^\lambda, \boldsymbol{A}, \boldsymbol{t})$, where $(\boldsymbol{A}, \boldsymbol{t})$ is the public verification key for Dilithium.
(2) $\mathcal{S}$ invokes $\mathcal{A}$ on inputs 1^λ and simulates oracle Π for $\mathcal{A}$ in $Expt - DistSign$ by the following steps:
   (a) $\mathcal{S}$ replies $\perp$ to all queries $(sid, \cdot)$ to Π by $\mathcal{A}$ before the key-generation subprotocol (3.1) is completed.
   (b) After $\mathcal{A}$ sends (0,0) to Π, simulator $\mathcal{S}$ receives $(0, message_1)$, which is P_1's first message in the key generation subprotocol. Simulator $\mathcal{S}$ computes

the reply as follows:

(i) $\mathcal{S}$ parses $message_1$ into the form $\left(\text{commit}, 1, \boldsymbol{t_1}, (\boldsymbol{s_{11}}, \boldsymbol{s_{12}})\right)$ that P_1 sends to $\mathcal{F}_{com-zk}^{R_{LWE}}$ in the hybrid model.

(ii) $\mathcal{S}$ verifies that $\boldsymbol{t_1} = \mathbf{A}\boldsymbol{s_{11}} + \boldsymbol{s_{12}}$ and $(\boldsymbol{s_{11}}, \boldsymbol{s_{12}}) \in S_\eta^k \times S_\eta^l$. If yes, then it computes $\boldsymbol{t_2} = \boldsymbol{t} - \boldsymbol{t_1}$, where $\boldsymbol{t}$ is received as the verification key in the experiment $Expt - Sign_{\mathcal{S},\pi}$; if no, then $\mathcal{S}$ just chooses a random $\boldsymbol{t_2}$.

(iii) $\mathcal{S}$ returns $(zk - proof, 2, \boldsymbol{t_2})$ and internally sends this to $\mathcal{A}$.

(c) Upon $\mathcal{S}$ receiving the next message $(0, message_2)$, it proceeds as follows:

(i) $\mathcal{S}$ parses $message_2$ into two messages: (1) $(Decommit, sid||1)$ as $\mathcal{A}$ intends to send to $\mathcal{F}_{com}^{R_{LWE}}$, and (2) $\left(zk - proof, 1, \left(pk_{BFV}, evk_{BFV}\right), sk_{BFV}\right)$ as $\mathcal{A}$ intends to send to $\mathcal{F}_{zk}^{R_{BFV}}$.

(ii) Similarly, $\mathcal{S}$ generates the oracle response to be P_2 aborting if $\boldsymbol{t_1} \neq \mathbf{A}\boldsymbol{s_{11}} + \boldsymbol{s_{12}}$ or $\boldsymbol{s_{11}} \notin S_\eta^k$. Because P_1 is corrupted, the simulator knows $(\boldsymbol{s_{11}}, \boldsymbol{s_{12}})$ and sk_{BFV}.

(iii) If $\mathcal{S}$ simulates an abort, then the experiment terminates. $\mathcal{S}$ does not output anything in this case because no verification key vk is output by P_2 in this case.

(iv) If $\mathcal{S}$ did not abort, then it stores $\left((\boldsymbol{s_{11}}, \boldsymbol{s_{12}}), \boldsymbol{t}, sk_{BFV}\right)$ and the distributed key generation phase is completed.

(d) When $\mathcal{S}$ receives a message of the form (sid, m), where sid is a new session identifier, $\mathcal{S}$ queries its signing oracle in experiment $Expt - Sign$ with message m and obtains a signature $(\boldsymbol{z}, c)$. Then, $\mathcal{S}$ processes the queries from $\mathcal{S}$ with identifier sid as follows:

(i) Upon receiving the first message $(sid, message_1)$, $\mathcal{S}$ parses $message_1$ as $(commit - prove, sid||1, \boldsymbol{W}_1, \boldsymbol{y}_1)$. If $\boldsymbol{W}_1 = A\boldsymbol{y}_1$ and $\boldsymbol{y}_1 \in S_{\gamma_1 - 1}^l$, $\mathcal{S}$ set $\boldsymbol{W}_2 = (\boldsymbol{Az} - c\boldsymbol{t}) - \boldsymbol{W}_1$; else it chooses $\boldsymbol{W}_2$ at random. $\mathcal{S}$ replies to $\mathcal{A}$ with the message $(proof, sid||2, \boldsymbol{W}_2)$. Noted that $\mathcal{S}$ simulates $\boldsymbol{Ay}$ using $(\boldsymbol{Az} - c\boldsymbol{t})$, which is feasible since $\boldsymbol{Az} - c\boldsymbol{t} = \boldsymbol{Ay} - c(\boldsymbol{s_{12}} + \boldsymbol{s_{22}})$ and the random oracle assumption.

(ii) On receiving the random oracle query for $(HighBits(\boldsymbol{W}_1 + \boldsymbol{W}_2, 2\gamma_2)||M)$, the simulator $\mathcal{S}$ programs the output as c and sends it to the adversary $\mathcal{A}$.

(iii) On receiving the second message $(sid, message_2)$ from $\mathcal{A}$, $\mathcal{S}$ parses $message_2$ as $(decommit, sid||1)$ and $\boldsymbol{c_1}$. If $\boldsymbol{W}_1 \neq A\boldsymbol{y}_1$ then $\mathcal{S}$ simulates P_2 aborting and the experiment completed. Otherwise, $\mathcal{S}$ computes $\boldsymbol{z_2} = \boldsymbol{z} - (\boldsymbol{y}_1 + c\boldsymbol{s_{11}})$, where $(\boldsymbol{z}, c)$ is the value from the signature received from $Expt - Sign$ if $\|\boldsymbol{z_2}\|_\infty \geq \gamma_1 - \beta$ or $\|LowBits(\boldsymbol{Az} - c\boldsymbol{t}, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$ then returns $\perp$; otherwise, $\mathcal{S}$ computes $\boldsymbol{c_4} = Enc_{pk}^{BFV}(\|LowBits(\boldsymbol{Az} - c\boldsymbol{t}, 2\gamma_2)\|_\infty)$, compute $\boldsymbol{c_5} = Enc_{pk}^{BFV}(HighBits(\boldsymbol{Az} - c\boldsymbol{t}, 2\gamma_2))$. Finally, $\mathcal{S}$ computes $c_2 = Enc_{pk_{BFV}}(z_2)$ and replies to $\mathcal{A}$ with the ciphertext c_2, c_4, c_5.

(3) Whenever $\mathcal{A}$ halts and outputs a pair (m^*, σ^*), simulator $\mathcal{S}$ outputs (m^*, σ^*) and halts.

We proceed to demonstrate that Eq. (1) holds. First, we prove that $\mathcal{A}$'s view in the simulation of key generation by $\mathcal{S}$ is statistically close to its view in a real execution of protocol 3.1. Because $\mathcal{S}$ defines $\boldsymbol{t_2} = \boldsymbol{t} - \boldsymbol{t_1}$, the public verify key in the $Exp - DistSign$ is defined to be $\boldsymbol{t_1} + \boldsymbol{t_2} = \boldsymbol{t_1} + (\boldsymbol{t} - \boldsymbol{t_1}) = \boldsymbol{t}$. Therefore, in our simulation, the public verification key generated by $\mathcal{S}$ with adversary $\mathcal{A}$ equals the public verification key $\boldsymbol{t}$ that it received in experiment $Exp - Sign$. Then, $\mathcal{A}$ outputs a pair (m^*, σ^*) that is a valid signature with the same probability in the simulation and in $Exp - DistSign$. Because the public verification key in the simulation is same as the public verification key in the $Exp - Sign$, a valid forge signature generated by $\mathcal{A}$ in $Exp - DistSign$ is a valid forgery by $\mathcal{S}$ in $Exp - Sign$; thus, Eq.(1) holds.

The difference between the simulation of key generation and a real execution with an honest P_2 is the way the public verification key is generated: P_2 samples random $(\boldsymbol{s_{21}}, \boldsymbol{s_{22}}) \in S_\eta^k \times S_\eta^l$ and $\boldsymbol{t_2} = \mathbf{A}\boldsymbol{s_{21}} + \boldsymbol{s_{22}}$, whereas $\mathcal{S}$ computes $\boldsymbol{t_2} = \boldsymbol{t} - \boldsymbol{t_1}$ where $\boldsymbol{t}$ is the public verification key received by $\mathcal{S}$ in $Expt - Sign$. Since $\boldsymbol{t}$ is randomly chosen, it follows that the distributions over $(\boldsymbol{t} - \boldsymbol{t_1})$ and $\boldsymbol{t_2}$ are identical. If P_2 does not abort, then the public verification key in both the simulation by $\mathcal{S}$ and a real execution equal $\boldsymbol{t_1} = \mathbf{A}\boldsymbol{s_{11}} + \boldsymbol{s_{12}}$. Furthermore, if $\boldsymbol{t_1} \neq \mathbf{A}\boldsymbol{s_{11}} + \boldsymbol{s_{12}}$ or $(\boldsymbol{s_{11}}, \boldsymbol{s_{12}}) \notin S_\eta^k \times S_\eta^l$, $\mathcal{S}$ simulate P_2 aborting. In a real execution, P_2 aborts in such a case if the zero-knowledge proof R_{LWE} fails. However, if $\boldsymbol{t_1} \neq \mathbf{A}\boldsymbol{s_{11}} + \boldsymbol{s_{12}}$ or $(\boldsymbol{s_{11}}, \boldsymbol{s_{12}}) \notin S_\eta^k \times S_\eta^l$, then by the soundness of zero-knowledge proof, P_2 aborts in real execution except with negligible probability. Therefore, we can conclude that the view of $\mathcal{A}$ in the simulation is statistically close to a real execution, and the output is public verification key $(\boldsymbol{A}, \boldsymbol{t})$.

Next, we prove that $\mathcal{A}$'s view in the simulation for the signing phase by $\mathcal{S}$ is statistically close to its view in a real execution of protocol 3.2. Observed that $\boldsymbol{W_2}$ is identically

distributed in both cases because $\boldsymbol{W}$ is randomly generated in the real signature generation; and thus $\boldsymbol{W} - \boldsymbol{W}_1$ has the same distribution as $\boldsymbol{W_2}$, where $\boldsymbol{W}$ is simulated as $\boldsymbol{Az} - c\boldsymbol{t}$. The zero-knowledge proofs and verifications are identically distributed in the $\mathcal{F}_{zk}$ and $\mathcal{F}_{com-zk}$ hybrid model. Thus, the only difference between the view of $\mathcal{A}$ in a real execution and in the simulation is that c_2, c_4 *and* c_5 are chosen.

In the simulation c_2 is the encryption of $\boldsymbol{z} - (\boldsymbol{y}_1 + c\boldsymbol{s_{11}})$, whereas in a real execution it is the encryption of $\boldsymbol{z}_2 = \boldsymbol{y}_2 + c\boldsymbol{s_{21}}$. We state that c_2 is identically distributed in both cases because $\boldsymbol{z}$ is randomly generated in the standard signature generation.

In the simulation c_4 and c_5 is the encryption of $\|LowBits(\boldsymbol{Az} - c\boldsymbol{t}, 2\gamma_2)\|_\infty$ and $LowBits(\boldsymbol{Az} - c\boldsymbol{t}, 2\gamma_2)$ respectively, whereas in an real execution of it is an encryption of $\|LowBits(\boldsymbol{A}(\boldsymbol{y_1} + \boldsymbol{y_2}) - c(\boldsymbol{s_{12}} + \boldsymbol{s_{22}}), 2\gamma_2)\|_\infty$ and $LowBits(\boldsymbol{A}(\boldsymbol{y_1} + \boldsymbol{y_2}) - c(\boldsymbol{s_{12}} + \boldsymbol{s_{22}}), 2\gamma_2)$. Because of the CPA security of the BFV scheme, the distributions of c_4 and c_5 in a real execution is as in the simulation.

Case 2: P_2 corrupted, $b = 2$. Let $\mathcal{A}$ be a PPT adversary in $Expt - DistSign^{b=2}_{\mathcal{A},\Pi}$. We will construct a PPT simulator $\mathcal{S}$ for $Expt - Sign_{\mathcal{S},\pi}$. This simulation is similar to the case where P_1 is corrupt, with some differences, i.e., the generation method of c_2, c_4 and c_5. These ciphertexts may be maliciously constructed by adversary $\mathcal{A}$, and the simulator cannot detect this. Therefore, simulating c_2, c_4 and c_5 controlled by corrupted P_2 is a problem. Let $q(\lambda)$ is an upper bound on the number of signing queries made $\mathcal{A}$ to Π in the experiment $Expt - DistSign$. Assume that $\mathcal{S}$ chooses a random $i \in \{1,2,\dots,q(\lambda)\}$, then $\mathcal{S}$'s choice of i is correct with probability $\frac{1}{q(\lambda)}$, i.e., $\mathcal{S}$ simulates $\mathcal{A}$'s view with probability $\frac{1}{q(\lambda)}$. Thus, if $\mathcal{A}$ forges a signature in $Expt - DistSign$ with advantage $\varepsilon(\lambda)$, $\mathcal{S}$ can forge a signature in $Expt - Sign$ with probability at least $\frac{1}{q(\lambda)} \cdot \varepsilon(\lambda)$. $\mathcal{S}$ should proceed as follows:

(1) The simulator $\mathcal{S}$ receives $(1^\lambda, \boldsymbol{t})$ in $Expt - Sign$, where $\boldsymbol{t}$ is the public verification key for Dilithium.

(2) Let $q(\lambda)$ is an upper bound on the number of signing queries that $\mathcal{A}$ makes to Π in the experiment $Expt - DistSign$. Assume that $\mathcal{S}$ chooses a random $i \in \{1,2,\dots,q(\lambda)\}$.

(3) $\mathcal{S}$ invokes $\mathcal{A}$ on input 1^λ and simulates oracle Π for $\mathcal{A}$ in $Expt - DistSign$, answering queries as follows:

(a) $\mathcal{S}$ replies $\perp$ to all queries $(sid, \cdot)$ to Π by $\mathcal{A}$ before the key-generation subprotocol is completed. $\mathcal{S}$ replies $\perp$ to all queries from $\mathcal{A}$ before it queries $(0,0)$.

(b) After $\mathcal{A}$ sends $(0,0)$ to Π, simulator $\mathcal{S}$ sends $(proof - receipt, 1)$ to $\mathcal{A}$.

(c) On receiving the next message of form $(0, message_1)$, $\mathcal{S}$ processes as follows:

(i) $\mathcal{S}$ parses $message_1$ into the form $(prove, 2, \boldsymbol{t_2}, s_{22})$ that P_2 sends to $\mathcal{F}_{com}^{R_{LWE}}$ in the hybrid model.

(ii) $\mathcal{S}$ verifies that $\boldsymbol{t_2} = \boldsymbol{As_{21}} + \boldsymbol{s_{22}}$; if not, it simulates P_2 aborting and halts.

(iii) $\mathcal{S}$ generates a valid BFV key-pair (pk_{BFV}, sk_{BFV}).

(iv) $\mathcal{S}$ returns the messages $(decom - proof, 1, \boldsymbol{t_1})$ and $(proof, 1, pk_{BFV})$, where $\boldsymbol{t_1} = \boldsymbol{t} - \boldsymbol{t_2}$ with $\boldsymbol{t}$ as received by $\mathcal{S}$ in experiment $Expt - Sign$ initially.

$\mathcal{S}$ stores $((\boldsymbol{s_{21}}, \boldsymbol{s_{22}}), \boldsymbol{t})$ and the key distribution phase is completed.

(d) On receiving a query of the form (sid, m) where sid is a new session identifier, $\mathcal{S}$ computes the oracle reply to be $(proof - receipt, sid||1)$ and hands it to $\mathcal{A}$.

Then, $\mathcal{S}$ queries its signing oracle in experiment $Expt - Sign$ with m and receives back a signature $(\boldsymbol{z}, c)$. $\mathcal{S}$ computes $HighBits(\boldsymbol{Az} - c\boldsymbol{t}, 2\gamma_2)$ and $LowBits(\boldsymbol{Az} - c\boldsymbol{t}, 2\gamma_2)$ using the Dilithium verification procedure. Then, $\mathcal{S}$ processes the queries from $\mathcal{A}$ with identifier sid as follows:

(i) Upon receiving the first message $(sid, message_1)$, $\mathcal{S}$ parses $message_1$ as $(prove, sid||2, \boldsymbol{w_2}, \boldsymbol{y_2})$ that $\mathcal{A}$ sends to $\mathcal{F}_{zk}^{R_{SIS}}$. $\mathcal{S}$ verifies that $\boldsymbol{W_2} = \boldsymbol{Ay_2}$ and that $y_2 \in S_\eta^k$; otherwise, it simulates P_1 aborting. Then, $\mathcal{S}$ set $\boldsymbol{W}_1 = (\boldsymbol{Az} - c\boldsymbol{t}) - \boldsymbol{W}_2$; else it chooses $\boldsymbol{W}_1$ at random. $\mathcal{S}$ replies to $\mathcal{A}$ with the message $(proof, sid||2, \boldsymbol{W}_2)$ as if coming from $\mathcal{F}_{zk}^{R_{SIS}}$.

(ii) On receiving the second message $(sid, message_2)$, $\mathcal{S}$ parses $message_2$ as c_2, c_4, c_5. If this is the ith call by $\mathcal{A}$ to the oracle Π, $\mathcal{S}$ simulates P_1 aborting; otherwise, it continues.

(4) Whenever $\mathcal{A}$ halts and outputs a pair (m^*, σ^*), simulator $\mathcal{S}$ outputs (m^*, σ^*) and halts.

$\mathcal{S}$ simulated the interaction in experiment $Expt - DistSign$ with $\mathcal{A}$. The public key generated by $\mathcal{S}$ in the simulation equals the public verify key $\boldsymbol{t}$ that received in the experiment $Expt - Sign$. Now, let j be the first call to oracle Π with (sid, c_2, c_4, c_5) where c_2, c_4, c_5 are such that P_1 does not get a valid signature $(\boldsymbol{z}, c)$ with respect to $\boldsymbol{t}$. Then, we argue that if $j = i$, the only difference between the distribution over $\mathcal{A}$'s view in real execution and the simulated execution by $\mathcal{S}$ is the ciphertext c_1. Indeed, $c_1 = Enc_{pk_{BFV}}(\boldsymbol{Ay}_1 - c\boldsymbol{s}_{11})$ in a real execution, whereas in the simulation $c_1 = Enc_{pk_{BFV}}(\tilde{s})$ for a random $\tilde{s}$ and is independent of $\boldsymbol{Ay}_1 - c\boldsymbol{s}_{11}$. We can see that $\mathcal{S}$ does not use

the private key for BFV at all in the simulation. Thus, the indistinguishability of this simulation follows from a straightforward reduction to the indistinguishability under CPA security of the BFV scheme, which proves that:

$$\left| \begin{matrix} Pr\left[Expt - sign_{\mathcal{S},\pi}\left(1^{\lambda}\right) = 1 | i = j\right] \\ -Pr\left[Expt - DistSign^{2}_{\mathcal{A},\Pi}\left(1^{\lambda}\right) = 1\right] \end{matrix} \right| \leq \varepsilon(1^{\lambda}),$$

and so $Pr\left[Expt - DistSign^{2}_{\mathcal{A},\Pi}\left(1^{\lambda}\right) = 1\right]$

$$\leq \frac{Pr\left[Expt - sign_{\mathcal{S},\pi}\left(1^{\lambda}\right) = 1 \wedge i = j\right]}{\Pr[i = j]} + \varepsilon(1^{\lambda})$$

$$\leq \frac{Pr\left[Expt - sign_{\mathcal{S},\pi}\left(1^{\lambda}\right) = 1\right]}{1/\mathrm{q}(\lambda)} + \varepsilon(1^{\lambda}).$$

And so $Pr\left[Expt - sign_{\mathcal{S},\pi}\left(1^{\lambda}\right) = 1\right]$

$$\geq \frac{Pr\left[Expt - DistSign^{2}_{\mathcal{A},\Pi}\left(1^{\lambda}\right) = 1\right]}{\mathrm{q}(\lambda)} - \varepsilon(1^{\lambda}).$$

The above indicates that if $\mathcal{A}$ forges a signature in $Expt - DistSign^{2}_{\mathcal{A},\Pi}$ with non-negligible probability, $\mathcal{S}$ forges a signature in $Expt - sign_{\mathcal{S},\pi}$ with non-negligible probability; therefore, it contradicts the assumed security of Dilithium.

IV. Performance Analysis

We analyze the computational and communication complexity of the two-party signing protocol. P_1 requires a standard Dilithium signature, a single BFV encryption, and three BFV decryptions. In contrast, P_2 requires a standard Dilithium signature, two BFV encryptions, and three BFV homomorphic evaluations. Furthermore, P_1 requires a committed non-interactive zero-knowledge proof of knowledge for SIS, and P_2 requires a zero-knowledge proof of knowledge for SIS. Regarding communication complexity, the protocol has four rounds of communication with each direction comprising two rounds.

Table 1 shows that two-party ECDSA protocols in [5] and [6] have lower computational complexity than ours. In terms of communications, all protocols use 4 rounds for Signing, but our protocol need more communication bandwidth. Our protocol has no advantage in computational complexity and communication complexity, however our protocol can resist the attack of quantum computing.

TABLE I. Performance Comparation

protocol	computation complexity	communication complexity	resist quantum attack
two-party ECDSA [5]	P_1: 1S +1D P_2: 1S +1Ev	4 rounds	No
two-party ECDSA [6]	P_1: 1S+1D P_2: 1S+1E+2Ev	4 rounds	No
two-party Dilithium (Our work)	P_1: 1S+1E+3D P_2: 1S+2E+3Ev	4 rounds	Yes

Note: S denotes a standard signature, E denotes an encryption, D denotes a decryption, and Ev denotes homomorphic evaluation.

V. Conclusions

In this study, we describe how to build a distributed key generation and distributed signing protocol for Dilithium. Our protocol achieves coordinated two-party Dilithium signature based on BFV homomorphic encryption scheme. Based on the random oracle assumption, our protocol is proven to be secure against active adversaries corrupting one of the players. Furthermore, the protocol achieves resistance to quantum computing attacks. In the future work, we will present a threshold signing protocol for Dilithium.

References

[1] R. Gennaro, S. Halevi, H. Krawczyk, and T. Rabin, "Threshold RSA for dynamic and Ad-Hoc groups," in Proc. Eurpcrypt2008, Istanbul, Turkey, 2008, pp. 88-107.

[2] V. Shoup, "Practical threshold signatures," in Proc. Eurocrypt2000, Bruges, Belgium, 2000, pp. 207-220

[3] R. Bendlin and I. Damgård, "Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems," in Proc. TCC 2010, Zurich, Switzerland, 2000, pp. 201-208.

[4] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin, "Robust threshold DSS signatures," in Proc. Eurocrypt1996, Saragossa, Spain, 1996, pp. 354-371.

[5] Y. Lindell, "Fast secure two-party ECDSA signing," in Proc. Crypto2017, Santa Barbara, CA, USA, 2017, pp. 613-644.

[6] G. Castagnos, D. Catalano, F. Laguillaumie, et al, "Two-party ECDSA from hash proof systems and efficient instantiations," in Proc. Crypto2019, Santa Barbara, CA, USA, 2019, pp. 191-221.

[7] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, et al, "CRYSTALS-Dilithium," [Online]. Available: https://pq-crystals.org/dilithium/index.shtml.

[8] Y. Lindell, "Highly-efficient universally-composable commitments based on the DDH assumption," in Proc. Eurocrypt2011, Tallinn, Estonia, 2011, pp. 446-466.

[9] C. Gentry, "Fully homomorphic encryption using ideal lattices," in Proc. STOC2009, Bethesda, MD, USA, 2009, pp.169–178.

[10] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in Proc. Crypto2012. Santa Barbara, CA, USA, 2012, pp. 868–886.

[11] Z. Brakerski, C. Gentry, V. Vaikuntanathan, "(Leveled) Fully homomorphic encryption without bootstrapping," in Proc. of the 3rd Innovations in Theoretical Computer Science Conference, Cambridge Massachusetts, USA, 2012, pp.309–325.

[12] J.Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Report 2012/144, 2012. Available: http://eprint.iacr.org/2012/144.

[13] C. Gentry, A. Sahai, B. Waters, "Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based," in Proc. Crypto 2013, Santa Barbara, CA, USA, 2013, pp.75-92.

[14] L. Ducas, D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," in Proc. Eurocrypt2015, Sofia, Bulgaria, 2015, pp. 617–640.

[15] I. Chillotti, N. Gama, M. Georgieva, et al, "Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE," Journal of Cryptology, Vol. 33, No. 1, pp. 34-91, Jan. 2020.

[16] SEAL, Available: http://sealcrypto.org.

[17] B.Blanchet. "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif". Foundations and Trends in Privacy and Security, 1(1-2):1-135, October 2016. http://dx.doi.org/10.1561/3300000004.