Algorithms for Information Security and Privacy
Assignment 4: Pairing and Blockchain
Group 3

Jisu Ge(48206116), Role1
Tianyu Yang(37205123), Role 2: Discussion A
Yifei Chen(48206640), Role 2: Discussion B
Yuxi Liu(48196147), Role 3: Discussion C
Yana Garipova(48206555) Role 3: Discussion D

January 12, 2021

Task for Role 1 (1 person): Section 1.1 of [1]: why the warm-up protocol discussed during the class is not secure.

In the warm-up protocol, we discussed that a message M is signed by each of the senders' private keys, which are used to generate an aggregate signature by group addition. The signature is then verified by an aggregated public key, which is found by group addition of the senders' public keys.

$$\sigma_i \longleftarrow \mathrm{H}_0(m)^{sk_i}$$

$$\sigma = \sigma_1 + \sigma_2$$
$$apk := pk_1 + \mathrm{pk}_2$$

The protocol is vulnerable to attacks, similar to the rogue public key attack. The attacker registers a rogue public key $pk_2 := g_2^{\alpha} - pk_1 \in \mathbb{G}_2$. This allows it to sign some message $m \in \mathcal{M}$ by presenting the aggregate signature $\sigma := \mathrm{H}_0(m)^{\alpha}$, which verifies the aggregate of pk_1 and pk_2:

$$e(\sigma, g_2) = e(\mathrm{H}_0(m)^{\alpha}, g_2) = e(\mathrm{H}_0(m), g_2^{\alpha}) = e(\mathrm{H}_0(m), pk_1 + pk_2)$$

Task for Role 2 (1 – 2 persons) Please read the protocol presented at Section 3.1 [1].
Discussion A: Prove that the protocol is valid (we can always verify the signature).

To prove the protocol is valid, we can do a derivation from the left side of the formula to the right side:

$$e(\tilde{\sigma}, g_2) = e(\mathrm{H}_0(m_1), apk_1^{\rho_1}) \cdots e(\mathrm{H}_0(m_b), apk_b^{\rho_b})$$

Since we know that $\tilde{\sigma} = \sigma_1^{\rho_1} \cdots \sigma_b^{\rho_b}$, $\sigma \leftarrow \prod_{i=1}^{b} s_j$ and $s_j \leftarrow \mathrm{H}_0(m_i)^{a_i sk_i}$, we can get this:

$$e(\tilde{\sigma}, g_2) = (\sigma_1^{\rho_1} \cdots \sigma_b^{\rho_b}, g_2) = (\prod_{i=1}^{b} \mathrm{H}_0(m_i)^{a_i sk_i \rho_i}, g_2) = e(\mathrm{H}_0(m_1)^{a_1 sk_1 \rho_1}, g_2) \cdots e(\mathrm{H}_0(m_b)^{a_b sk_b \rho_b}, g_2)$$

From the properties of weil pairing, we can get this from the above formula:

$$= e(\mathrm{H}_0(m_1), g_2)^{a_1 sk_1 \rho_1} \cdots e(\mathrm{H}_0(m_b), g_2)^{a_b sk_b \rho_b} = e(\mathrm{H}_0(m_1), g_2^{a_1 sk_1 \rho_1}) \cdots e(\mathrm{H}_0(m_b), g_2^{a_b sk_b \rho_b})$$

Then since $pk \leftarrow g_2^{sk}$, and $apk \leftarrow \prod_{i=1}^{b} pk_i^{H_1(pk_i, \{pk_i, \cdots, pk_n\})} = \prod_{i=1}^{b} pk_i^{a_i}$, we can calculate that:

$$= e(\mathrm{H}_0(m_1), pk^{a_1 \rho_1}) \cdots e(\mathrm{H}_0(m_b), pk^{a_b \rho_b}) = e(\mathrm{H}_0(m_1), apk^{\rho_1}) \cdots e(\mathrm{H}_0(m_b), apk^{\rho_b})$$

In summary, we can always verify the signature, and we can say that this protocol is valid.

Discussion B: Give a reason why you think that this protocol is secure. (You do not need to read the proof in Section 3.2. Please just give your own intuition here.)

The MSP algorithm introduces two hash functions H_0 and H_1 into the aggregation, signing and verification processes. The hash function H_1 makes the signed message m distinct to allow the protocol become more secure
Let's take a quick review of the rogue-key attack in conventional BLS algorithm.
An attacker first picks a secret key a from the prime order q, and use generator g_2 to get a fake aggregated public key g_2^a. In the meantime, the attacker makes a fake aggregated signature, based on $H_0(m)^a$. These two are both generated by a, and this aggregated signature can be verified. The attacker knows the Bob public key pub_1, and attacker has a aggregated public key g_2^a. So, he could finally retrieve a rogue public key pub_2 such that $pub_1 \cdot pub_2 = g_2^a$. He could claim that the $H_0(m)^a$ is signed by me and Bob, and it will be successfully verified.

Differently from the methods above, the MSP algorithm modifies the BLS algorithm by adding a hash algorithm H_0 and H_1 in both aggregation and signing processes. The main goal is to make every signed message unique to separate with others. During signing, instead of signing on message $H_0(m)$, now the message we are signing with sk_i is actually $H_0(m)^{a_i}$, though $H_0(m)$ is the same, $a_i \leftarrow H_1(pk_i, \{pk_1, ..., pk_n\})$ is different for each member.

When the attacker has a rogue-public key, he could only generate a message m that is non-distinct because he doesn't know the hash function. Because hash function is a one-way function, as long as we let the verifier rejects an aggregate signature on non-distinct messages, the protocol will be secure.

Other features of their method includes supporting off-line mode, and there is no interaction between signers needed. This paper's method also supports batch verification, a set of multi-signatures can be verified as a batch faster than verifying them one group means MSP requires public key and signature to live in two different.

Task for Roles 1-2 Please discuss why the protocol in Section 3.1 is secure against the attack in Section 1.1. (Again, you do not need to read the proof. Please just give your own intuition.)

In MSP, the aggregated signature now is not simply put $H_0(m)^{sk_1}$, $H_0(m)^{sk_2}$, $H_0(m)^{sk_n}$... together. They now become $H_0(m)^{a_1 \cdot sk_1}$, $H_0(m)^{a2_2 \cdot sk_2}$, $H_0(m)^{a_n \cdot sk_n}$...
Because a_1,a_2,a_n..., are calculated by hash function $a_i \leftarrow H_1(pk_i, \{pk_1, ..., pk_n\})$ So, now the attacker cannot simply make a fake aggregate signature. He gets some rogue-key for example pub_1 and it will give him a unique a_1. Because hash function is a one-way function which will randomly mapping the input to output. Although $H_0(m)$ is the same, a_i is different for each member. Thus the message $H_0(m)^{a_i}$ will be distinct.
It is similar to the second defense mechanic in Section 1.1. We let the verifier rejects an aggregate signature on non-distinct messages. This is sufficient to prevent the rogue-key attack. We make each message distinct. Because we use the hash function H_1 to generate a_i and apply a_i on the message m. We don't have to change the message m manually anymore compare with the second defense mechanic in Section 1.1. The message will become distinct during signing process.

Task for Role 3 (1 – 2 persons) Please read Section 4.2.
Discussion C: Please discuss what is Accountable-Subgroup Multisignatures (ASM) using your own words.

The target of multisignature is to generate the aggregate public key and signature that pass the verification. If we simply multiply the individual signatures as in Section 1.1, this will allow the rogue public key attack.

In ASM, we multiply each signature with the membership key mk_i.
The signature $s_i \leftarrow \mathrm{H}_0(apk, m)^{sk_i} \cdot mk_i)$ can be thought of as consisting two parts, the usual signature for the message $\mathrm{H}_0(apk, m)^{sk_i}$, and a signature that indicates the membership mk_i. The membership key prevents the attacker from rogue public key attack, playing a similar role as a_i in the *MSP* protocol of Section 3.

The verification procedure should perform two functions in correspondence: one is to verify the message is signed $e(\mathrm{H}_0(apk, m), pk)$, the other is to verify the membership $e(\prod_{j \in S} \mathrm{H}_2(apk, j), apk)$. The signature should pass both of these verifications, thus the verification condition is:

$$e(\mathrm{H}_0(apk, m), pk) \cdot e(\prod_{j \in S} \mathrm{H}_2(apk, j), apk) \stackrel{?}{=} e(s, g_2)$$

To generate such a membership key, every member should exchange information that contains both the group's identity (represented by the aggregate public key apk) and its own identity (represented by its secret key sk_i). We can't obviously send the secret key to each other, instead, we send $\mu_{j,i} = \mathrm{H}_2(apk, j)^{a_i \cdot sk_i}$ to each other, where $a_i \leftarrow \mathrm{H}_1(pk_i, \mathcal{PK})$ is the component corresponding to i-th part of apk. Then, we get the membership key $mk_i \leftarrow \prod_{j=1}^{n} \mu_{i,j}$.
From the above analysis, we can understand the forward scheme easily:

1. Generate essential parameters.
2. Each member generates its own key pair (pk_i, sk_i).
3. Each member computes $apk \leftarrow \prod_{i=1}^{n} pk_i^{H_1(pk_i, pk_1, ... pk_n)}$
4. Each member generates $\mu_{j,i}$ and exchanges with each other, then computes its own membership key mk_i. This concludes the group setup.

5. When members of a subset S want to sign a message m, they sign it with $s_i \leftarrow \mathrm{H}_0(apk, m)^{sk_i} \cdot mk_i$.
6. The combiner computes the aggregate public key $pk \leftarrow \prod_{j \in S} pk_j$ and the aggregate signature $s \leftarrow \prod_{j \in S} s_j$.
7. If pk and s pass the verification, then we know that members in S indeed sign the message m.

Discussion D: Please discuss the protocol in the section in your own words and class notations.

To describe the ASM protocol in class notation, let us recall the Weil Pairing properties. In the article, it is defined with the multiplication as the binary operation:

$$e(S_1 \cdot S_2, T) = e(S_1, T) \cdot e(S_2, T)$$
$$e(S, T_1 \cdot T_2) = e(S, T_1) \cdot e(S, T_2)$$

However in the class, we use addition as the group binary operation and define the Weil Pairing properties like so:

$$e(S_1 + S_2, T) = e(S_1, T) \cdot e(S_2, T)$$
$$e(S, T_1 + T_2) = e(S, T_1) \cdot e(S, T_2)$$

Additionally, the exponential operations become multiplication notation.
With notations being defined, let us discuss the ASM protocol on the example of 3 parties.

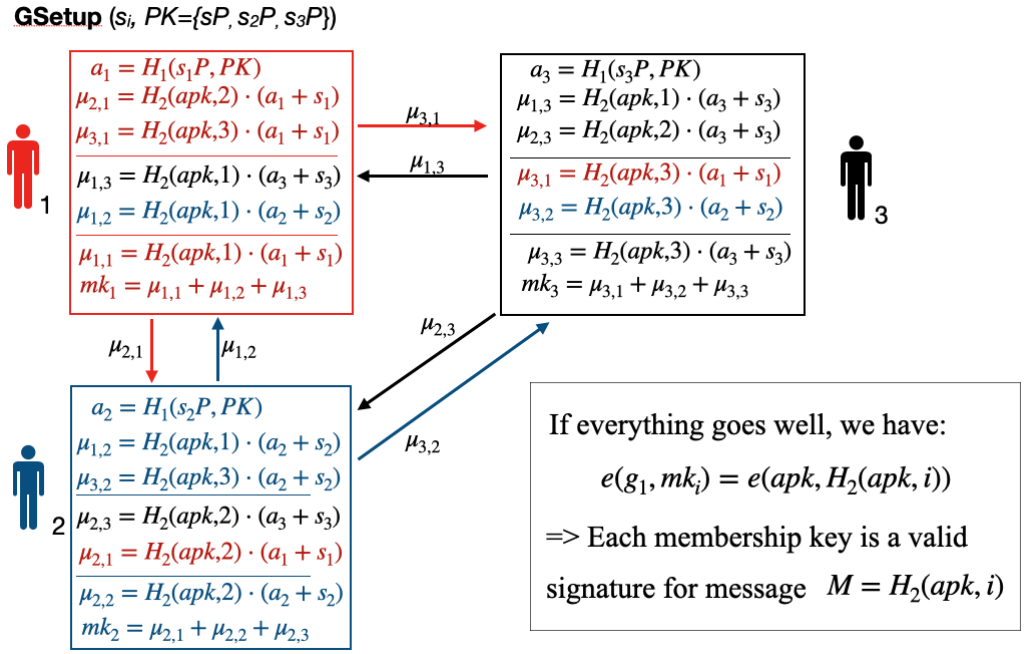
ASM is a protocol that allows one or more people to sign a message on behalf of a larger group. As the name states, ASM uses multisignatures to sign messages while shrinking the amount of data written to the blockchain. As described in the discussion C, the goal of the protocol is to verify the signers using the compact aggregate public key while being secure against the rogue public key attack.

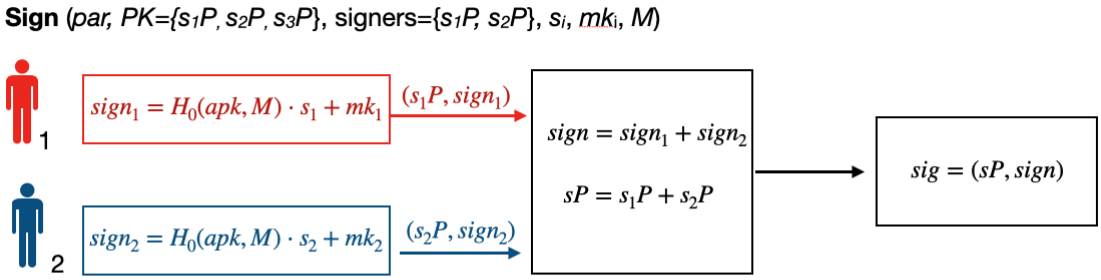
Assume we have a group of 3 people in a secret society and we want to use ASM protocol to allow only 2 members of the group to sign messages on behalf of the third person as well. As mentioned previously, the ASM protocol consists of 6 main steps with execution of Pg, Kg, GSetup, Sign, KAg, Vf algorithms, which are simply summarized in this table:

#	Algorithm	Input	Function	Output
1	**Pg**	--	Initialize common system parameters *par*	*par*
2	**Kg**	*par*	Generates a key pair for each signer (*public_key, private_key*)	(*sP, s*)
3	**KAg**	*group_set*	Creates aggregate public key *apk* from group set	*apk*
4	**GSetup**	*sP, group_set*: indexed collection of public keys, *apk*	- Ran by each group member to create a personal membership key *mk*. - Hash functions	*mk*
5	**Sign**	*par, group_set*, signers subset, *sP, mk*, message, *apk*	- Ran by subset of signers to create multisignature *sig* - Hash functions	*sig*
6	**Vf**	*par, apk*, signers subset, message, *sig*	Validation	0 or 1

As the first steps of the protocol, we generate starting parameters with Pg, then generates a key pair (s_iP, s_i) for each member. The indexed collection of public keys forms the group set PK and is used to find the aggregated public key apk with KAg algorithm.

GSetup is the next step that creates a membership key. Here, every group member creates a temporary hash value μ and shares it with every other member. Once a person received all the μ values from other members, he/she calculates his/her's personal μ and uses them all to form the membership key.





Vf(par, apk, signers={s_1P, s_2P}, M, sig) == *1 when:*

$$e(H_0[apk, M], sP) \cdot e(H_2[apk, s_1P] + H_2[apk, s_2P], apk) = e(sig, g_2)$$

In the end, we can verify whether certain signer-members of the group signed the message.

Conclusion

In this assignment we have discussed protocols developed in [1] that use signature compression and public-key aggregation to optimize the space usage in a blockchain. Based on the classic Schnorr and BLS-signatures, the authors developed new protocols MSP and ASM, which are efficient, secure, easy to use and require no proof of key ownership.

References:

[1] Neven G., Drijvers M., Boneh D. (2018). Compact Multi-Signatures for Smaller Blackchains. Stanford University.