The Theory Behind Blockchains (Spring 19) June 2019

Problem Set 5 - Solution

Submitted by: RW

1 Unique Signatures

a. For a given (known to be) image y, there exist (at least) one $x \in \mathbb{Z}_N^*$ s.t. $y = x^e \bmod N$.

For $i \in [p]$, element x_i is defined as:

$$x_i = x \cdot a^{i-1} \bmod N$$

Claim 1 *The following holds:*

$$x_1^e \equiv x_2^e \equiv ... \equiv x_p^e \pmod N$$

Proof: To prove this claim, it is enough to show that $x_1^e \bmod N = x_i^e \bmod N$ for any $i \in [p]$. It is known that $a^p = 1 \bmod N$, and that p divides e. Thus, $a^e = 1 \bmod N$ as well. From here:

$$x_i^e \bmod N = \left(x \cdot a^{i-1}\right)^e \bmod N = x^e \cdot (a^e)^{i-1} \bmod N = x^e \cdot (1)^{i-1} \bmod N = x^e \bmod N$$

which means that $x_1^e \bmod N = x_i^e \bmod N$ and thus the claim holds and there is a single image for the described elements. ■

Claim 2 *The following holds:*

$$x_1 \neq x_2 \neq ... \neq x_p$$

Proof: Assume there exist $i \neq j$ ($i, j \in [p]$) s.t. :

$$x \cdot a^{i-1} \bmod N = x \cdot a^{j-1} \bmod N$$

If the above phrase holds, $a^{i-1} \bmod N = a^{j-1} \bmod N$ holds as well.
w.l.o.g assume $i < j$, which means that $a^{j-i} = 1 \bmod N$.
It must hold that $j - i < p$, which contradicts p being the smallest integer such that $a^p = 1 \bmod N$.
Thus, the assumption must be wrong, i.e. $x_i \neq x_j$ for any $i \neq j$, $i, j \in [p]$ and the claim holds. ■

In total, for any given image p distinct matching elements can be found in the described manner and the mapping is p-to-1.

b. The protocol for a proof system in which a prover will attempt to prove that a given public key $p_k \in \{0,1\}^n$ represents a function $F_{p_k} : \{0,1\}^n \to \{0,1\}^n$ that is $\frac{1}{n}$-close to a permutation is as follows, where β is the security parameter:

   - The verifier chooses a random $y \in \{0,1\}^n$ and sends it to the prover.
   - After recieving y, the prover computes $x = F_{p_k}^{-1}(y)$ and sends it to the verifier. If there is no such value, it sends $\perp$.
   - After recieving x, the verifier validates that indeed $y = F_{p_k}(x)$. If x is $\perp$ or the validation fails, the verifier rejects.
   - The above steps are repeated for β iterations. After finishing these iterations without rejection, the verifier accepts.

   In case F_{p_k} is a permutation, in every iteration the prover will be able to "invert" y and compute the value of x for which $y = F_{p_k}(x)$, as the function must be surjective. Thus, the verifier will accept in this case.

   In case the size of F_{p_k}'s image is at most $(1 - \frac{1}{n})2^n$, in every iteration there is a probability of at least $\frac{1}{n}$ that the prover will not be able to "invert" y, as there at least $\frac{1}{n}2^n$ values out of the possible 2^n values in $\{0,1\}^n$ for which no element is mapped to. The verifier validates the mapping afterwards, and thus the prover can't profit by cheating.
   The probability that the verifier will accept in that case is no more than $(\frac{n-1}{n})^\beta$, as the prover will manage to pass each iteration in a probability which is less or equal to $\frac{n-1}{n}$. Thus, the probability that the verifier will reject is at least $1 - (\frac{n-1}{n})^\beta$.
   In order to make the verifier reject in (at least) some constant probability p, the security parameter will be set to:

$$\beta = \log_{\frac{n-1}{n}}(1-p)$$

c. The protocol for a proof system in which a prover will attempt to prove that a given public key $p_k \in \{0,1\}^n$ represents a function $F_{p_k} : \{0,1\}^n \to \{0,1\}^{2n}$ that is $\frac{1}{n}$-close to a injective is as follows, where β is the security parameter:

   - The verifier chooses a random $x \in \{0,1\}^n$, computes $y = F_{p_k}(x)$ and send y to the prover.
   - After recieving y, the prover computes $x' = F_{p_k}^{-1}(y)$ and sends it to the verifier.
   - After recieving x', the verifier validates that indeed $y = F_{p_k}(x')$. If $x' \neq x$ or the validation fails, the verifier rejects.
   - The above steps are repeated for β iterations. After finishing these iterations without rejection, the verifier accepts.

   In case F_{p_k} is an injective, in every iteration the prover will be able to "invert" y and to x only, as it is the only element for which y is mapped since the function is an injective. Thus, the verifier will accept in this case.

In case the size of F_{p_k}'s image is at most $(1-\frac{1}{n})2^n$, by the Generalized Pigenhole Principle there are at least $\frac{1}{n}2^n$ elements which are mapped to a value for which other elements are mapped to as well. Thus in every iteration there is a probability of at least $\frac{1}{n}$ that the element chosen by the veirifer (x) is one of those.
In such scenario, as the prover only recieves the mapped value, it will send one of at least two equally mapped elements back to the verifier. The probability that the prover will be able to cheat and guess is no more than $\frac{1}{2}$ - the more elements mapped to y, the less probable the guess of x is. Thus, in each iterations the probability that the verifier will reject is at least $\frac{1}{2n}$.
The probability that the verifier will accept is no more than $(\frac{2n-1}{2n})^{\beta}$, as the prover will manage to pass each iteration in a probability which is less or equal to $\frac{2n-1}{2n}$. Thus, the probability that the verifier will reject is at least $1-(\frac{2n-1}{2n})^{\beta}$.
In order to make the verifier reject in (at least) some constant probability p, the security parameter will be set to:

$$\beta = \log_{\frac{2n-1}{2n}}(1-p)$$

2 Algorand's Consensus

a. Given said variant of Algorand, it is assumed that at the begining of round r the value of $N(r)$ is known to the attacker (as it has joined the system before the start of round r) and that any nodes that publish their public key during round r will be candidates to be in the committee of round $r+1$.

   Any node that joins the system during round r (or before) will have a probability of $\frac{n}{N(r)}$ to be in the committee for round $r+1$, as the hash result of its signature on r should be in the first $\frac{n}{N(r)} \cdot 2^n$ values out of the possible 2^n values, while the hash result is uniformly distributed as it is modeled as a random oracle.

   In order to try and gain an entrance to the committee, the attacker can publish a new public key (as a new "fake" node if needed) - by generating a large enough of key-pairs, the probabiliy that at least one of them will fulfill the requirment to join the committee should be sufficient.
   By generating β key pairs and signing with each private key on r, the probability that at least one hash result (the lowest) will be low enough to gain an entrance to the committee of round $r+1$ is:

$$P_{\beta} = 1-(1-\frac{n}{N(r)})^{\beta} = 1-(\frac{N(r)-n}{N(r)})^{\beta}$$

   Hence, given a probability the required number of pairs generated should be:

$$\beta = \log_{(\frac{N(r)-n}{N(r)})}(1-P_{\beta})$$

   In total, by generating $\beta = \log_{(\frac{N(r)-n}{N(r)})} 0.01$ key pairs and publishing the public key with the lowest hash result before the start of round $r+1$, the attacker has a probability of 0.99 to be on the committee. The running time of said procedure scales with n and $N(r)$.

b. As Algorand assumes at least $\frac{2}{3}$ honest stakes, in order to take control of the entire system it is sufficient for the attacker to control a third or more of a specific committee to control the system.
Starting from round r, the attacker will first participate regularly in the system until one of the nodes she controls will be chosen as a leader of round $r + j$.
As for the committee membership assignments, the probability that a player will be chosen as the leader of a round is equal to its stake. Thus, the expected number of rounds until one of her nodes is chosen as a leader of a round is 10. The probability that it would take more than 100 rounds is 0.00003 and hence it is safe to assume that no more than expected number of rounds will pass before one the attacker nodes will become a leader.

Before round $r + j$, the attacker should choose the value of S_{r+j+1} which will be used to determine the participants of the committee. As the attacker controls tenth of the nodes, while each node has the same probability to be in the committee, the probability that at least a third of the committee will consist of the attacker nodes depends on n only and not on $N(r)$. In a simillar manner to the previous attack, for a given high probability for success (say 0.99) the number of iterations (where the attacker will attempt to find a S_{r+j+1} for which there are at least $\frac{n}{3}$ attacker nodes in the committee) can be calculated. In the almost certain case where a relevant S_{r+j+1} was found, the leader will publish the value and thus the attacker will control the system by "breaking" the committee.

In total, both phases of the attack should take up-to a constant number of steps, with a very high probability to successfully gain control of the system.

c. The previous attack relies on using the value of S that determines which nodes are in the committee of the next round in order to ensure control of it. When the value is calculated using the hash on the previous value and the former leader signature, it is now random and can't be predicted by an attacker. The previous attack will not be able to iterate through different values of S for the round after the attacker leads for usage in any way.

Even when the attacker is a leader for a round, she is committed to the relevant public key she has used to register the node to the system and will get a uniformly distributed value. The attack would work in case the attacker could be chosen as a leader in the first round after publishing a public key, as it would know how to choose it correctly to ensure control of the committee afterwards. The restriction on the leader candidates to n rounds back makes the attacker unable to do anything but guess the hashes until it may be chosen as a leader, which reduces the probability of her profiting from doing so to a negligible probability of the form $2^{-\Omega(n)}$ as the hash function is modeled as a random oracle.

In total, in the new situation the probability that the described attack (or some other attack of simillar principles) will allow the attacker to gain control of the committee is negligible.

3 Anonymous Coins

Based on the relevant lecture, recitation, Zerocoin & Zerocash white papers.

a. As described in the description, the zero knowledge proof π and the serial S are sent together with transaction destination public key p_k, without any signing on p_k. Thus, it can be replaced in the transaction with any other public key p'_k without invalidating it. In case the modified transaction will enter the blockchain first, it will persist and invalidate the original transaction, effectively stealing the coins.

   In order to prevent such scenario, one-time signature scheme $(Gen, Sign, Ver)$ can be used to augment zerocoin. The minting of one zerocoin will now be done by the steps:

   - Generate serial number S and a random secret r.
   - Generate a public and private keys for the one-time signature scheme by $p^o_k, s^o_k \leftarrow Gen(1^n)$.
   - Compute $Commit(S, p^o_k, r)$, a commitment to the serial number and the public one-time signature scheme public key.
   - Publish the commitment onto the block chain.

   In order to spend that zerocoin and create a transaction to some public key p_k, the following steps should be followed:

   - Generate a NIZK proof π of the statement: “I know r such that $Commit(S, p^o_k, r)$ is in the set $\{C_1, ..., C_n\}$”.
   - Calculate the one-time signature on $p_k : \sigma = Sign_{s^o_k}(p_k)$.
   - Publish the transaction $(S, p^o_k, \pi, p_k, \sigma)$ to the blockchain.

   After recieving a transaction, a node will verify it by:

   - Verify that π is indeed valid for S and p^o_k.
   - Verify that $Ver_{p^o_k}(p_k, \sigma) = 1$.

   The manner in which the one-time signature scheme was used ensures that a transaction destination can only be chosen by the node who minted the relevant committed coin.
   By embedding the one-time signature shceme public key into the commitment, the node will be able to publish the public key in a verifiable manner using the zero knowledge proof, still without revealing the specific commitment. The private key will be used once to sign on the public key of transaction destination, thus from the scheme being one-time an attacker will not be able to forge a signature (negligible

probability).
Since the public key will be published in the transaction itself (next to the destination and signature), other nodes will be able to verify the validity of the signature before accepting it into their blockchain and thus the transaction is now not modifiable by some other node.

b. In the Zerocoin protocol, the zero knowledge statement that each node has to prove in order to spend a minted coin is "I know r such that $Commit(S, r)$ is in the set $\{C_1, ..., C_n\}$". where S is the serial it publishes together with the proof and $\{C_1, ..., C_n\}$ are all the previous commitments in the blockchain.
The witness in this case should be a proof-of-membership of $Commit(S, r)$ in that set. In the paper, this proof-of-membership is done by a one-way accumulator. The accumulator is global, available in each block to describe the set of all minted coins so far. Using the accumulated value of the entire set (available in the blockchain) and of the set without the specifed commitment, a proof-of-membership of a single minted coin is in constant size, publicly computable and verifiable efficiently. It can be seen that the witness is indeed small (constant size), in contrast to the statement which requires all previous commitments and thus scales with their amount.

In order to guarantee that the statement will only scale logarithmically in the amount of all coins, Merkle trees can be used instead of the accumulated values in said proof-of-membership. Append-only Merkle tress of all published commitments so far will be maintained along the blockchain, with the updated root included in each published block. Using an efficient implementation, updating said value can be done in time/space proportional to the tree depth and thus logarithmically in the amount of all leaves (commitments). More importantly to our requirment, in order to prove that a commitment is indded included in the Merkle tree, hashes of each sibling up the tree starting from the representative leaf are required, totalling to the tree depth. Together with the publicly available root, the Merkle path is sufficient to prove that commitment is indded published and belongs to the set.

Thus, the new statement will be "I know r such that $Commit(S, r)$ appears as a leaf in the Merkle tree whose root is rt", where S is the serial published with the proof and rt is the publicly available root of the Merkle tree that contains all commitments so far. The witness in that case will be the Merkle path which proves that the commitment indeed appears as a leaf in the Merkle tree, while the statement will require all these relevant siblings up the tree.

In total, both the statement and the witness will scale logarithmically in the amount of all coins when using Merkle trees in the specified manner.