CS2362 - Project Report

Archisman Dutta

April 2023

1 Goals

The goals of the project are as follows:

- Motivate and define cryptographic accumulators as a decentralized alternative to digital signatures
- Show a basic construction of a collision-resistant RSA accumulator, as proposed by Benaloh and de Mare, and rigorously prove its security
- Implement a rudimentary version of a local decentralized anonymous membership testing system that is able to validate membership in a storage-constrained environment without (1) requiring the involvement of a trusted authority and (2) revealing the identities of the individual members

2 Motivation

Assume there is a clandestine collective of honest individuals that wish to authenticate the identity of each member without revealing any information to a non-member. The most obvious approach would be to maintain a membership list in the form of a set X given by:

$$X := \{a_1, a_2, \ldots, a_n\}$$

where $a_1, a_2, \ldots, a_n$ consist of the personal identifying information of all n members. Thus, anyone in the collective can search for the presence of the

element a_i in X to authenticate the identity of the i^{th} member. However, this is not feasible in a storage-constrained environment since it requires providing copies of the membership list to every individual. Moreover, it would also need to be ensured that no one leaks their copy of the list to any non-member.

An alternate approach could involve recruiting a trusted centralized authority to digitally sign an identity card for each member using its own private key and publish the corresponding public key for identity verification. Thus, at a later stage, any member's identity could be verified using their signed identity card and the trusted authority's public key. However, this would entail trusting said authority not to engage in any kind of forgery or fraud, which might not be suitable in certain zero-trust threat models.

Cryptographic accumulators can be utilized as a foolproof decentralized measure in this situation. First, a succinct representation (or digest) z of the aforementioned set X is computed that reveals nothing about the value of any element $a \in X$. This can then be used to verify $a \in X$ by computing a membership witness w of all the elements in $X \setminus \{a\}$ and showing $z = h(w, a)$ where h is a suitable accumulator function.

3 Definition

A cryptographic accumulator is a tuple of probabilistic polynomial-time (PPT) algorithms $(\mathsf{Gen}, \mathsf{Acc} : X \to A, \mathsf{Wit} : X \times M \to W, \mathsf{Ver} : X \times W \times A \to \{\mathsf{Member}, \mathsf{NotMember}, \mathsf{Error}\})$ such that A is the accumulated value domain and W is the membership witness domain with the following conditions:

- **Generating algorithm** $\mathsf{Gen}(1^k)$ accepts as input the security parameter 1^k and the set M, and outputs an auxiliary value aux
- **Accumulator algorithm** $\mathsf{Acc}_{\mathsf{aux}}(X)$ accepts as input the set $X \subseteq M$ and outputs the accumulated value a of X as a short representation of X
- **Witness generation algorithm** $\mathsf{Wit}_{\mathsf{aux}}(e, X)$ accepts as input the set $X \subseteq M$ and an element $e \in M$ and outputs a membership witness $w_e \in W$

- **Verification algorithm** $\mathsf{Ver}_{\mathsf{aux}}(e, w_e, a)$ accepts as input the accumulated value a, the element $e \in M$ and the membership witness w_e, and outputs either Member or $\mathsf{NotMember}$ or Error.

- **Correctness**: $\mathsf{Ver}_{\mathsf{aux}}(e, \mathsf{Wit}_{\mathsf{aux}}(e, X), \mathsf{Acc}_{\mathsf{aux}}(X))$ outputs Member if $e \in X$, $\mathsf{NotMember}$ if $e \notin X$, and Error otherwise for valid $\mathsf{aux} \in \mathsf{Gen}(1^k), X \subseteq M, x \in M$.

- **Security**: There exists a negligible function $\mathsf{negl}(k)$ for all $k \in \mathbb{N}$, for all M and for all PPT adversaries A such that:

$$\mathbb{P}[\mathsf{aux} \in \mathsf{Gen}(1^k); (X^*, e^* \in M, w_{e^*}, w'_{e^*}) \in A(\mathsf{aux}) \mid \mathsf{Ver}(e^*, w_{e^*}, a)$$

$$= \mathsf{Member} \wedge \mathsf{Ver}(e^*, w'_{e^*}, a) = \mathsf{NotMember}] \leq \mathsf{negl}(k)$$

4 Construction

Benaloh and de Mare construct a hash-based one-way RSA accumulator as a function $h : X \times Y \to X$ that satisfies the following property of quasi-commutativity:

$$h(h(x, y_1), y_2) = h(h(x, y_2), y_1)$$

The one-wayness of such functions imply that, given an input pair $(x, y) \in X \times Y$ and a $y' \in Y$, it is difficult to compute an $x' \in X$ such that $h(x, y) = h(x', y')$. In other words, there is no polynomial P for which, given some sufficiently large integer l, there exists a probabilistic polynomial-time algorithm that can find an $x' \in X_l$ with probability greater than $\frac{1}{P(l)}$ such that $h_l(x, y) = h_l(x', y')$ for a given $(x, y) \in X_l \times Y_l$ and $y' \in Y_l$ sampled uniformly at random with $|X| \approx |Y|$. However, for every integer l, $h_l(x, y)$ can be computed in polynomial time $P(l, |x|, |y|)$ given $x \in X_l$ and $y \in Y_l$.

Starting with an initial value $x \in X$ and a set of values $y_1, y_2, \ldots, y_m \in Y$, one can calculate the accumulated hash z as follows:

$$z = h(h(h(\ldots h(h(h(x, y_1), y_2), y_3), \ldots, y_{m-2}), y_{m-1}), y_m)$$

The order of all y_i, $1 \leq i \leq m$, can be permuted without modifying the accumulated hash z because of its quasi-commutative nature. This makes it possible for users of a cryptosystem to calculate the partial accumulated

hash (or membership witness) z_j of all y_i with $i \neq j$ and validate their own y_j by verifying $z = h(z_j, y_j)$. Forgery is impractical because anyone wishing to validate a fake y' would need to compute an x' such that $z = h(x', y')$. This is infeasible for any given function that is dependent on an underlying hardness assumption for some intractable mathematical problem like discrete logarithm computation.

The modular exponentiation function $e : X \times Y \to X$, $e_n(x, y) = x^y \bmod n$, which is quasi-commutative for all n. That is, for any given $x \in X$ and $y_1, y_2 \in Y$,

$$e_n(e_n(x, y_1), y_2) = (x^{y_1})^{y_2} \bmod n = (x^{y_2})^{y_1} \bmod n = e_n(e_n(x, y_2), y_1)$$

For all appropriately chosen n, it is difficult to compute an $x \in X$ in time polynomial in $|n|$, given the values of $e_n(x, y), y, n$, assuming root finding modulo n is hard for all such n.

5 Security

To be used as one-way accumulators such that collision resistance is preserved despite repeated applications of e_n that reduce the size of the image, it is necessary to restrict the domain of n to rigid integers only. Formally, $n = pq$ such that $|p| = |q|$ where $p = 2p' + 1$ and $q = 2q' + 1$, p' and q' are odd primes. Prime factorization is assumed to be a computationally hard problem that underlies the security parameter of these one-way functions. The group of quadratic residues modulo n that are coprime to n have a size n' coprime to y where $n' = \frac{(p-1)}{2}\frac{(q-1)}{2}$ and the function $e_n(x, y) = x^y \bmod n$ is a permutation of this group. So, arbitrary exponentiations of this group will produce elements of a proper subgroup with negligible probability. It follows that repeated applications of $e_n(x, y)$ will produce collisions with negligible probability as long as the factorization of n is unknown since root finding modulo n is supposed to be computationally difficult.

Even if there exists a polynomial time algorithm A that is provided with the roots $y_1, y_2, \ldots, y_k$ and the indices $r_1, r_2, \ldots, r_k$, where each $y_i^{r_i} \bmod n = x$, and is able to find a y for a given r such that $y^r \bmod n = x$, Benaloh and de Mare demonstrate that there exists a polynomial time algorithm B which does not require the roots $y_1, y_2, \ldots, y_k$ to find y such that $y^\rho \bmod n = x$,

given x, n and $\rho = r/\gcd(r, r_1, r_2, \ldots, r_k)$. Since root finding is an intractable problem, the roots $z^{1/r_1}, z^{1/r_2}, \ldots, z^{1/r_k}$ are insufficient for computing $z^{1/\rho}$ unless ρ divides $\prod_{i=1}^{k} r_i$ where $z = x^y \bmod n$. An r^{th} root for a given $z \bmod n$ can be computed only if given a set of known roots and indices $\{(x_i, r_i : x_i^{r_i} \bmod n = z\}$ where r divides $\prod r_i$. Even if the adversary is allowed to pre-select all root indices, for some n of the order of magnitude 10^{220}, Benaloh and de Mare mention that around 20 million items can be hashed such that forgery would happen with a negligible probability far less than 10^{-30}.

6 Implementation

The one-way accumulator, as proposed by Benaloh and de Mare, is utilized to develop a simple anonymous membership testing system in Python that can validate user membership without needing the intervention of a trusted authority or revealing user identities.

Let the membership of m users for some system be given in the form of values $y_1, y_2, \ldots, y_m$ and an initial x be decided upon. The accumulated hash value z can be calculated using these values as

$$z = h(h(h(\ldots h(h(h(x, y_1), y_2), y_3), \ldots, y_{m-2}), y_{m-1}), y_m)$$

Now, each user has access to the one-way hash function h and can compute the membership witness z_j of all y_i such that $1 \leq i \leq m$ and $i \neq j$. The membership of any valid user can be authenticated using their respective value y_j by computing $z = h(z_j, y_j)$ and no user needs to know about the membership of any other user since the knowledge of z is sufficient to verify that some user's own value was indeed used in the construction of the accumulated hash and the collision resistance of one-way accumulators make it infeasible to engage in membership forgery. The details of the implementation are as follows:

- In the GitHub repository linked below, there are three separate programs ComputeAccumulator.py, GenerateWitness.py and VerifyMembership.py.
- The first program ComputeAccumulator.py takes as input a file containing the identities of members, in this case, their email IDs. For demon-

stration purposes, a list of randomly generated sample email IDs are used.

- Each of these IDs is hashed to a unique prime in a deterministic manner using SHA256.
- Two 256-bit safe primes p and q are randomly generated and their product n is computed.
- A generator g of the prime order cyclic group $\mathbb{Z}_n^*$ is arbitrarily selected (since all elements in $\{2, \ldots, n-1\}$ are generators).
- The accumulated value A is calculated by computing the product p of all the prime numbers that the member email IDs are mapped to, and then calculating $g^p \bmod n$ since the modular exponentiation function is quasi-commutative as shown earlier.
- The second program GenerateWitness.py takes as input the value of the rigid integer, the generator and the accumulator value alongside the file containing the member identities.
- Membership witnesses are generated for all the members in the form of the generator raised to the product of all the aforementioned prime numbers (apart from the respective prime at the given index) which can then be distributed to the respective members for membership verification later.
- The third program VerifyMembership.py takes as input the value of the rigid integer, the accumulator value as well as the membership witness and email ID of the member whose identity is to be verified.
- The membership witness is raised to the power of the prime number corresponding to the email ID of the member and membership is verified if the output matches the accumulator value.

Find the repository here.

7 References

Benaloh, J., de Mare, M.: One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In: Helleseth, T., ed. EUROCRYPT 1993.

LNCS, vol. 765, pp.274–285, Lofthus, Norway, Springer, Heidelberg, 1994 (23–27 May 1993)
Lipmaa, H. (2012). Secure Accumulators from Euclidean Rings without Trusted Setup. In: Bao, F., Samarati, P., Zhou, J. (eds) Applied Cryptography and Network Security. ACNS 2012. Lecture Notes in Computer Science, vol 7341. Springer, Berlin, Heidelberg
Tiwari, P., Jhawar, M.: Trading Accumulation Size for Witness Size: A Merkle Tree Based Universal Accumulator via Subset Differences.