Cryptographic Accumulators and their Application in Anonymous Blacklisting

Hooman Mohajeri Moghaddam

David R. Cheriton School of Computer Science
University of Waterloo, Ontario, Canada

Abstract. Cryptographic accumulators, initially introduced in the 90's, have recently regained popularity in privacy preserving communications. What makes these accumulators very practical, is the fact that a user can prove a statement about a value he or she possesses, with regard to a universal value known to everyone (the accumulator value). This property combined with the new zero-knowledge proof techniques, enable us to perform authentications without revealing users' identities, a very useful technique in so called *anonymous blacklisting* schemes. We have done a survey of current cryptographic accumulators and how they are used in anonymous blacklisting schemes, implemented an instance of *modular exponentiation* construction and compared it under different settings, and we are presenting the results here.

1 Motivation

In most online services where the common client-server setting is used, users initially sign up with the service provider and are given credentials in return, for further authentications. Each user's credential is a proof that the client holding that credential is indeed the same user who signed up initially. Therefore, in order for a user to access the system, he needs to reveal his identity to the service provider. Although this approach is widely used, there are cases where we need to be able to authenticate users, but at the same time preserve their privacy. A very good example which illustrates why such a system is required, is anonymous edits and contributions to Wikipedia. It seems obvious to require authentication when a user changes or edits some sensitive article, but at the same time, one can think of situations where a user is willing to contribute to an article only when his identity is kept secret, for example due to future threats and persecution. Hence an anonymous blacklisting (AB) system, where users can authenticate to service providers (SPs) as some anonymous member of a group [9], seems extremely useful for such cases. The main goal in such system is to be able to prevent a misbehaving user from accessing the service, but not allow anyone to link between users' activities. There have been a number of attempts to study and formalize this concept [5].

A very straight forward way of implementing AB systems is using Trusted Third Parties (TTP), where the TTP knows the identity of each user and grants authentication token only to non-misbehaving users. However, it is non-trivial

to implement a TTP-free AB system. PEREA[9] in particular is an anonymous blacklisting scheme which eliminates the need for a TTPs using a variety of cryptographic tools including Universal Dynamic Accumulator and Zero-knowledge Proof techniques, which we will describe next.

2 Background

The idea of cryptographic accumulators first appeared in the work of Benaloh and deMare [2]. The authors defined the notion of *quasi-commutative* functions:

Definition 1. *[2] A function $f : U \times X \to U$ is said to be quasi-commutative if for all $u \in U$ and all $x_1, x_2 \in X$ we have:*

$$f(f(u, x_1), x_2) = f(f(u, x_2), x_1) \tag{1}$$

This definition combined with the idea of one-way hash functions, where the inverse of a point in the image of f is hard to compute, results in the families of one-way accumulators:

Definition 2. *A family of one-way accumulators is a family of one-way hash functions each of which is quasi-commutative. [2]*

The quasi-commutative property is very useful in proving that a value is a part of an accumulator, by a statement such as what follows:

Imagine that parties $1, 2, ..., n$ calculate the following value:

$$v = h(h...h(h(u, x_1), x_2), ...x_{n-1}), x_n) \tag{2}$$

where x_i belongs to party i and h is a one-way accumulator. Since h is a quasi-commutative function, the order of evaluation does not matter and therefore any party can prove that his value is a part of the accumulator by presenting w_j such that $v = h(w_j, x_j)$. In this case v is called the accumulator value and w_j is called the *membership witness*[1] for accumulated value x_j.

Theorem 1. *Given a one-way accumulator h, for n parties, holding values $x_1, x_2, ..., x_n$, the accumulated value $v = h(h...h(h(u, x_1), x_2), ...x_{n-1}), x_n)$ and the witnesses for all users can be computed using $O(n \log n)$ evaluations of h.*

Proof. (This proof is done by the author): Accumulator value v can be easily calculated using $O(n)$ evaluations.

Lets assume $\mathcal{X} = \{x_1, x_2, ..., x_n\}$. To prove the $n \log n$ bound for witnesses, we generalize the concept of witness to a witness for more than one value. For simplicity consider the set of values $\{x_1, x_2, ..., x_k\}$ of values, define witness $w_{1,2,...,k}$ to be $w_{1,2,...,k} = h(...h(h(u, x_{k+1}), x_{k+2}), ..., x_n)$

[1] In general, by witness we mean membership witness unless otherwise stated.

To calculate witnesses, we construct a tree-like structure, where leaves correspond to witnesses for single values. Without loss of generality assume that $n = 2^k$ for some integer k (for the general case set $k = 1 + \log n$). We construct a binary tree from bottom to top. The leaves are w_1, w_2, ... , w_n. Then the parent of w_1 and w_2 is $w_{1,2}$. Similarly for any odd $i \leq n$ the parent of w_i and w_{i+1} is $w_{i,i+1}$. Following the same construction, for any depth, the parent of two adjacent witnesses $w_{\mathcal{X}_1}$ and $w_{\mathcal{X}_2}$ is $w_{\mathcal{X}_1 \cup \mathcal{X}_2}$, where $\mathcal{X}_1, \mathcal{X}_2 \subset \mathcal{X}$.

If the root of the tree is depth zero, then it is easy to see that to construct a child at depth i from a parent at depth $i-1$ we need to perform 2^{k-i} evaluations of h and since at depth i we have 2^i nodes in the tree, calculating each depth of the tree requires $n = 2^k$ evaluation of h. Given that the depth of the tree is $k = \log n$, the result is $n \log n$ evaluation of h. □

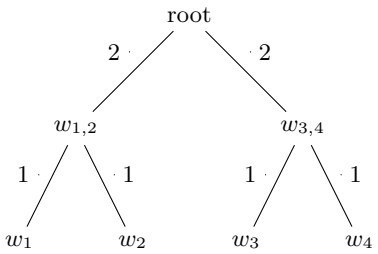

Fig. 1: Binary tree construction for four witness values. Values on the connecting edges show the number of evaluations needed to calculate the child from the parent.

Figure 1 shows an example of size 4.

Next we introduce the notion of security for an accumulator [1]:

Definition 3. *An accumulator construction is secure if given the accumulator value, and the set of accumulated values $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ with accumulator value v, it is difficult for an adversary to come up with a witness w for $x \notin \mathcal{X}$ such $h(w, x) = v$.*

The difficulty in the definition above usually means computation difficulty, i.e., there exists no probabilistic polynomial algorithm that succeeds with non-negligible probability.

3 Cryptographic Accumulators: Definitions

Here we collect some of the definitions introduced in order to formalize cryptographic accumulators. It is worth noting that our literature review shows that the main theme in the definition of all accumulators so far has been the same

basic idea of being able to combine different values into one unified value such that users can prove statements about this unified value. As we will see some constructions allow more capabilities, for example non-membership witnesses.

Definition 4. *[3]: A secure accumulator for a family of inputs $\mathcal{X}_k$ is a family of families of functions $\mathcal{G} = \mathcal{F}_k$ with the following properties:*

- *Efficient generation: An efficient probabilistic algorithm that on input k produces a function $f \in \mathcal{F}_k$, with some auxiliary information about f (e.g. factorization of N)*
- *Efficient evaluation: f can be evaluated in polynomial time.*
- *Quasi-commutative: For all k and all $f \in \mathcal{F}_k$, for all $x_1, x_2 \in \mathcal{X}_k$, we have $f(f(u, x_1), x_2) = f(f(u, x_2), x_1)$*
- *Security: It is difficult for an adversary to create a witness for a value not accumulated in the accumulator.*

Camenisch and Lysyanskaya [3] extend this definition to accommodate for deletion of values from the accumulator at a later time after initiation. They call the an accumulator with this capability a *dynamic accumulator*:

Definition 5. *[3]: A secure accumulator is dynamic if it has the following property: Efficient deletion: Given a set $\mathcal{X} = \{x_1, ..., x_n\}$, if $v = f(u, \mathcal{X})$,[2] there exists an efficient algorithm to compute the followings:*

1. *$v' = f(u, \mathcal{X} \backslash \{x_i\})$ for $x_i \in \mathcal{X}$.*
2. *Witnesses w'_j for all other values $x_j \in \mathcal{X}$ such that $v' = f(w'_j, x_j)$.*

To further extend this idea, Li et. al. proposed the idea of *universal dynamic accumulators* where user can provide non-membership witnesses:

Definition 6. *[6]: A dynamic accumulator is called universal, if users can provide non-membership witness for values not accumulated in the accumulator.*

4 Modular Exponentiation Construction

The first construction for cryptographic accumulators appeared in the work of Benaloh and de Mare [2], where they proposed using modular exponentiation as the basis for constructing accumulators. Their construction uses an RSA modulus N and a value $x \in \mathbb{Z}_N$ which is the initial value of the accumulator. Then for every value e to be accumulated, we raise the current value of accumulator to power of e. However, in their work, there is no formal definition of security.

Baric and Pfitzmann [1] extended this construction and gave a more rigorous proof of security by proposing the following assumption called *strong RSA assumption*:

[2] For a set $\mathcal{X} = \{x_1, ..., x_n\}$, the notation $f(u, \mathcal{X})$ means $f(...f(f(u, x_1), x_2, ..., x_n)$ for a quasi-commutative function f.

Definition 7. ***Strong RSA assumption*** *[1]: Given a randomly chosen RSA modulus N and $z \in_R \mathbb{Z}_N^*$, [3] it is difficult to find $r > 1$ and $y \in \mathbb{Z}_N^*$ such that $y^r = z \pmod N$.*

It is worth noting that, this assumption differs from the standard RSA assumption in that, the adversary is allowed to pick the exponent r of his choice in addition to y. However, there seems to be no efficient way to break the assumption without knowing the factorization of N. For example, if one choose r at random, then breaking strong RSA assumption will result in breaking normal RSA assumption. If y is chosen at random, then finding r is equivalent to finding a discrete logarithm of z. Also other attacks so far have failed to break the assumption; therefore it seems like a reasonable assumption to make.

Construction 1 *Assume p and q are primes and $N = pq$ is an RSA modulus and $\mathcal{X} = \{e \in \mathbb{Z}_N^* | \ e \ \text{is a prime}\}$. Construct the following accumulator function:*

$$f : \mathbb{Z}_N \times \mathcal{X} \to \mathbb{Z}_N$$
$$f(z, e) = z^e \pmod N$$

It is easy to see that $f(f(z, e_1), e_2) = f(f(z, e_2), e_1) = u^{e_1 e_2} \pmod N$, i.e., f is quasi-commutative. Also note that the elements in the set $\mathcal{X}$ are chosen to be prime which eliminate any correlation between any elements in the range of f. This is suggested by Shamir [7] for generating pseudorandom sequences based on one-way functions. However, this choice also allows for proving the security of this construction under the strong RSA assumption:

Theorem 2. *If the set of values to be accumulated is $\mathcal{X} = \{e \in \mathbb{Z}_N^* | \ e \ \text{is a prime}\}$ then the accumulator in construction 1 is secure (according to definition 3) under the strong RSA assumption.*

Proof. [1, 7] Given primes $e_1, e_2, ..., e_k$ accumulated in the value $z^{\bar{e}}$, where $\bar{e} = \prod_{i=1}^{k} e_i$, lets assume the adversary can compute values s and r such that $s^r = z^{\bar{e}}$ and r is a prime number. We will find r-th root of z which violates the strong RSA assumption.

Since r is prime and is not accumulated in the accumulator value, we have $\gcd(r, \bar{e}) = 1$. Now using Euclidean algorithm we can find $a, b \in \mathbb{Z}$ such that $a\bar{e} + br = 1$. Now set $y = s^a z^b$. The following shows that y is the r-th root of z:

$$y^r = (s^a z^b)^r = s^{ar} z^{br} = (z^{\bar{e}})^a z^{br} = z^{a\bar{e}+br} = z \pmod N$$

□

[3] $z \in_R \mathbb{Z}_N^*$ denotes choosing element z uniformly at random from $\mathbb{Z}_N^*$.

Nevertheless, note that anyone who knows the factorization of N can easily come up with a witness for a value that is not necessarily accumulated. To see this, consider the current accumulator value v and compute non-zero m and n such that $m \equiv n^{-1} \mod \Phi(N)$. Now setting $w = v^m$, one can prove that w is a witness that n is accumulated in v since $w^n = v^{mn} = v \mod N$.

Baric and Pfitzmann [1] also present a similar construction, using a *random oracle*, where values to be accumulated are first passed through the random oracle (outputing a random value) and then the accumulator value is raised to the resulting value. This construction is secure under the standard RSA assumption.

The work by Camenisch and Lysyanskaya [3] extends the previous construction and presents a *dynamic accumulator* where values can be added and deleted from an accumulator and they also provide a zero-knowledge protocol for proving that a committed value is in the accumulator (which we will not discuss here).

Definition 8. *A prime number p is called a* **safe prime** *if $p = 2p' + 1$ where p' is an odd prime (A number p' of this form is known as a Sophie Germain prime). [2]*

Now consider the RSA modulus $N = pq$ where p and q are safe primes. The subgroup QR_N of *quadratic residues* modulo N is a subgroup of order $p'q'$. To see this we use the following theorem:

Theorem 3. *Suppose $n > 1$ is an odd integer having factorization $n = \prod_{i=1}^{l} p_i^{e_i}$ where the p_i's are distinct primes and the e_i's are positive integers. Suppose further that $\gcd(a, n) = 1$. Then the congruence $y^2 \equiv a \pmod{n}$ has 2^l solutions modulo n if $(\frac{a}{p_i}) = 1$ for all $i \in \{1, ..., l\}$, and no solutions otherwise.*[4] *[8]*

Using theorem 3 it is easy to see the following mapping is a four-to-one mapping, i.e., there are four points in the preimage of each value in the range of $\mathcal{M}$:

$$\mathcal{M} : \mathbb{Z}_N \to \mathbb{QR}_N$$
$$\mathcal{M}(z) = z^2 \mod N$$

Now choosing u at random from QR_N. If $u \neq 1$, with high probability $< u >= QR_N$ or $< u >$ is large enough, since the order of u is either p',q' or $p'q'$. This allows for generating sequences with large periodicity. Thus Camenisch and Lysyanskaya [3] proposed the following construction:

Construction 2 *Assume p and q are safe primes and $N = pq$ is an RSA modulus and QR_N is the subgroup of quadratic residues modulo N. Further assume $\mathcal{X} = \{e \in \mathbb{Z}_N^* | \ e \ \text{ is a prime}\}$. Construct the following accumulator function:*

$$f : QR_N \times \mathcal{X} \to QR_N$$
$$f(u, e) = u^e \mod N$$

[4] $(\frac{a}{p_i})$ is used to denote the Legendre symbol. For more detail, see [8].

The above construction is similar to construction 1, the only difference is the choice of QR_N as the range of accumulator, thus the security proof of theorem 4 works for this construction as well. However, the authors took advantage of the trapdoor information for a faster batch calculation of witnesses and also efficient deletion of accumulated values.

To see how one can use the trapdoor information for creating witnesses for a set of values, consider the set of primes $\mathcal{E} = \{e_1, e_2, ..., e_k\}$ to be accumulated. It is straightforward to see that given $t = \prod_{i=1}^{k} e_i$ and the accumulator value $v = u^t$, the witness for value e_i is $v^{\bar{e_i}}$ where $\bar{e_i} = e_i^{-1} \pmod{\Phi(N)}$. Therefore, once the accumulator value is calculated, each witness can be computed using a modular inverse computation followed by a single modular exponentiation.

Now to see why the modular exponentiation construction is a dynamic accumulator, i.e., we can efficiently delete values from the accumulator, note that we can remove a value e_j from the accumulator by computing the new accumulator value $v' = v^{e_j^{-1} \pmod{\Phi(N)}}$ since we know $\Phi(N) = (p-1)(q-1)$.

Proposition 1. *The witness for value e_i after deletion of value e_j is $w_i' = v'^a {w_i}^b$ where a and b satisfy $ae_i + be_j = 1$.*

Proof. [3] Since e_i and e_j are both primes, using Euclidean algorithm we can find a and b such that $ae_i + be_j = 1$ holds. Now we prove that $(w_i')^{e_i} = v'$ also holds, i.e., w_j' is a new witness for e_j:

$$\begin{aligned}
(w_i')^{e_i} &= (v'^a {w_i}^b)^{e_i} \\
&= ((v'^a {w_i}^b)^{e_j e_i})^{1/e_j} \\
&= ((v'^{e_j})^{ae_i} ({w_i}^{e_i})^{be_j})^{1/e_j} \\
&= (v^{ae_i} v^{be_j})^{1/e_j} \\
&= v^{1/e_j} = v'
\end{aligned}$$

□

We can also prove that modular exponentiation is universal accumulator or members can provide a non-membership witness for values not accumulated in the accumulator value v. A non-membership witness for a value e in this case is a tuple (r, s) such that $v^r = s^e u \pmod{N}$. Again, it is difficult to forge a non-membership proof under strong RSA assumption:

Theorem 4. *Assume that in construction 2, the set $\mathcal{X} = \{e_1, e_2, ..., e_k\}$ is accumulated in $v = u^t$, where $t = \prod_{i=1}^{k} e_i$. Then the adversary cannot find a non-membership witness for any $e \in \mathcal{X}$ under the strong RSA assumption.*

Proof. If the adversary can compute a tuple (r, s) such that $v^r = s^e u \pmod{N}$ then we show he can break the RSA assumption by computing y such that $y^e = u$. First we know that $v^r = s^e u \pmod{N}$ and $v = u^t$, thus:

$$u^{tr-1} = s^e \tag{3}$$

Since $e \in \mathcal{X}$ we have $e \mid t$ and so $\gcd(tr-1, e) = 1$. Calculate a and b using Euclidean algorithm such that $a(tr-1) + be = 1$. Now set $y = s^a u^b$. Then we have:

$$\begin{aligned} y^e &= (s^a u^b)^e \\ &= (s^e)^a u^{be} \\ &= u^{a(tr-1)} u^{be} = u \end{aligned}$$

□

4.1 On prime generation algorithms

The modular exponentiation introduced in this section requires two prime generation algorithms, one for key generation (determining the RSA modulus) and one for the values accumulated in the accumulator. We have implemented and used two algorithms suggested by Cramer and Shoup [4].

Key Genereation :

In order to generate the RSA modulus, we need to generate primes $p = 2p' + 1$ such that p' is again a prime number. Even though the number of primes less than or equal to x is proportional to $\frac{x}{\ln(x)}$, the number of Sophie Germain is conjectured to be proportional to $\frac{x}{(\ln(x))^2}$. Therefore, we use the following algorithm proposed in the work of Cramer and Shoup [4]:

1. Generate a random, odd number p' of desired length.
2. Test if either p' or $p = 2p' + 1$ are divisible by any primes up to some bound B. If so, go back to step 1.
3. Test if 2 is a Miller-Rabin witness to the compositeness of p'. If so, go back to step 1.
4. Test if $2^{p'} \equiv \pm 1 \pmod{p}$. If not, go back to step 1.
5. Apply the Miller-Rabin test to p' some number t times using randomly selected bases to ensure an error probability of ϵ. A reasonable choice of ϵ is $\epsilon = 2^{-80}$.

This algorithm yields a factor of 10 speed-up over the naive method of generating Sophie Germain primes by selecting p' and then checking if $2p' + 1$ is prime.

Prime generation for accumulation :

For generating primes used for accumulating values, we use the following algorithm (which was initially used for producing 160-bit primes) that outputs a guaranteed prime numbers relatively fast. Lets assume we need l bit primes. We first generate a random prime P of size $\lfloor \frac{l}{3} \rfloor - 1$ bits (this primality test can be done fast since the bit size is small enough). Then repeatedly an integer R is chosen in the interval $((2^l - 1)/2P, (2^{l+1} - 1)/2P)$ until $e = 2PR + 1$ is prime. Using the following theorem we can check the primality of e very efficiently:

Theorem 5. *[4] Let e, P, and R be as above. Then e is prime if and only if the following conditions hold:*

(i) There exists an integer a such that
- $a^{e-1} \equiv 1 \mod e$, *and*
- $\gcd(a^{2R} - 1, e) = 1$.

(ii) If $R = 2Px + y$; where x and y are integers with $0 \leq y < 2P$, then $y^2 - 4x$ is neither 0 nor a perfect square.

(iii) $R \not\equiv m \mod 2Pm + 1$ for all integers m with $1 \leq m < e/4P^3$.

We have implemented these two prime generation algorithm and compared them to the naive method in the experiment section.

5 Implementation and Evaluation

We have implemented the dynamic accumulator in construction 2 using Maple Software, and ran our experiment using a dual core 64-bit Intel CPU with clock rate of 2.40 GHz and 4.00 GB of RAM (The source code and the result files are included in the package sent by this report). We tweaked the following five parameters and reported the results:

- **Modulus size**: The size of RSA modulus N used for the accumulator. The size of modulus is particularly important since as previously stated, if the modulus can be factored, the accumulator will no longer be secure. We use the algorithm explained in section 4.1 for key generation.
- **Number of Additions**: The number of accumulation done after initialization. The addition is done one at a time, since it is easy to see that batch addition only requires additional modular multiplication and therefore can be done pretty fast.
- **Number of Deletions**: Similar to addition, the number of deletion is varied to see the effect on the time requirements.
- **Size of Primes**: The size of primes used for accumulation. If this size is small, the adversary may try to find the set of accumulated values. Although this is not a very powerful attack, it may be practical for lower prime sizes.
- **Prime Generation Algorithm**: We compare two algorithms for generating primes used for accumulation. The first one is the naive method of choosing a random number in the specified range and iterate through values larger than that number to find a prime of desired size. The other algorithm is the one explained in section 4.1.

During each experiment, we vary only one parameter and keep the others unchanged to make sure others parameters would not affect the experiment. The time reported here is the total time for the corresponding setting. For example in the first experiment, the total time for adding 200 values and removing 50 primes of size 100 bits, across different moduli is reported. Also each experiment is repeated 10 times and the average and the standard deviation are reported as figures.

5.1 Results

Modulus size: This parameter seems to be the dominating factor in the performance of an accumulator. As seen in figure 2, as the size of modulus gets larger, the time required to evaluate the value of accumulator grows rapidly. However, since the key generation algorithm used is randomized, the variance is also large. Also the key generation is a one-time operation and can be done offline, therefore, it should not be the bottleneck in a system.

```
Setting:
Modulus size := Variable
Prime Size := 100
Number of Adds := 200
Number of Deletes := 50
Fast Prime Generation := false
```

Modulus Size	256	512	768	1024
Average Time (seconds)	9.66	45.59	181.75	506.26
Standard deviation	1.68	23.08	82.16	337.64

Table 1: Time versus modulus size

Number of Additions As suggested by construction 2, addition of new values to the accumulator, scales linearly. This is shown in figure 3.

```
Setting:
Modulus size := 512
Prime Size := 100
Number of Addition:= Variable
Number of Deletes := 50
Fast Prime Generation := false
```

Number of Deletions Again as suggested by construction 2, deletion of values from the accumulator, scales linearly. This is shown in figure 4. Note

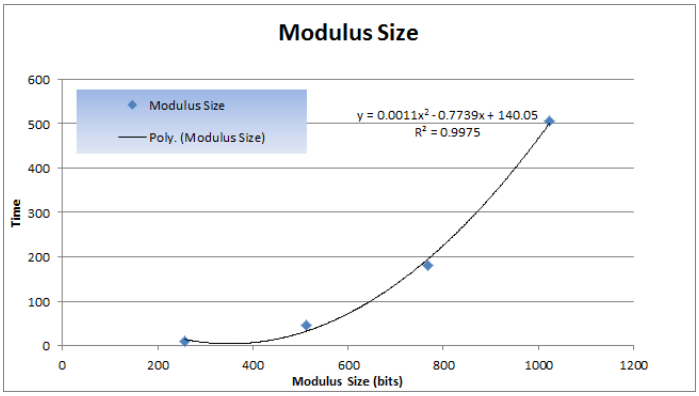

Fig. 2: Time versus modulus size

Number of additions	200	400	800	1600	3200
Average Time (seconds)	9.38	20.39	45.45	130.30	429.74
Standard deviation	1.62	3.19	3.63	12.78	56.75

Table 2: Time versus number of additions

that the concavity of the graph in the figure is probably due to the fact that as we delete more values, the number of witnesses to be updated decreases and consequently the rate of time required to calculate the witnesses also decreases.

```
Setting:
Modulus size := 512
Prime Size := 100
Number of Addition:= 800
Number of Deletes := Variable
Fast Prime Generation := false
```

Number of deletions	50	100	200	400	800
Average Time (seconds)	132.72	174.01	236.21	307.39	400.94
Standard deviation	31.81	34.85	24.93	20.50	37.54

Table 3: Time versus number of deletions

Prime Size and Fast Prime Generation

To see the effect of prime sizes on the time requirements, we have plotted two curves in figure 5. One of the curves shows the peformance of the fast prime

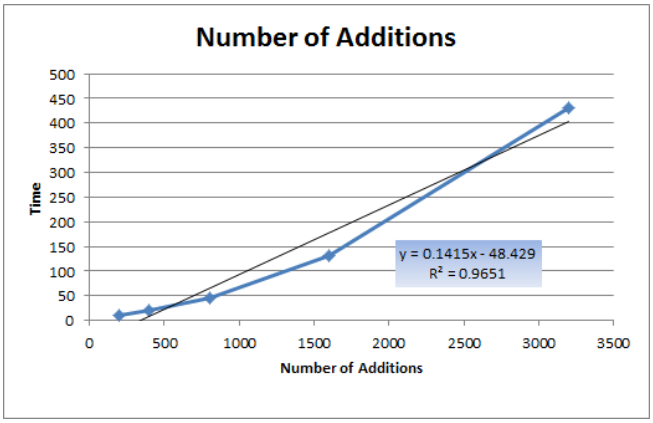

Fig. 3: Time versus number of additions

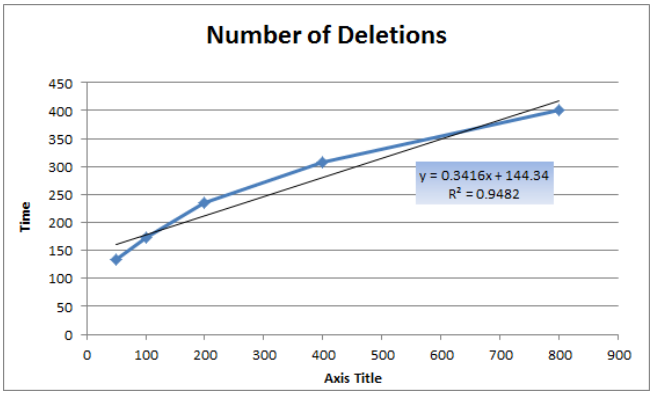

Fig. 4: Time versus number of deletions

generation algorithm discussed in section 4.1, the other one shows the naive method of selecting a random integer and finding the next prime bigger than that number. As the graph suggests, the fast prime generation method performs better for values less than 200 bits. This range conforms with the value 160 for which the algorithm was used in the work of Cramer and Shoup [4]. As it turns out, for values greater than 200 bits, the naive method should be used.

```
Setting:
Modulus size := 512
Prime Size := Variable
Number of Addition:= 800
Number of Deletes := 100
Fast Prime Generation := Variable
```

Prime Size	100	150	200	250	300
Fast Prime Generation Method Average Time in seconds	50.71	63.99	62.99	87.53	94.54
Normal Prime Generation Method Average Time in seconds	59.59	72.01	80.07	70.08	92.38

Table 4: Time versus size of primes

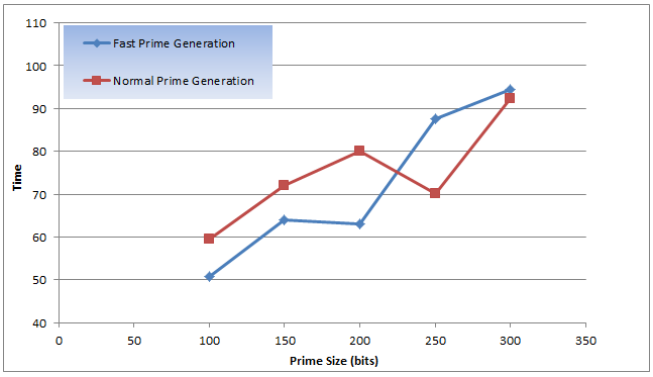

Fig. 5: Time versus size of primes

6 Anonymous Blacklisting Scheme

Now we turn to the application of cryptographic accumulators. One drawback of these accumulators is that a witness for a value can only be revealed to a trusted party, otherwise that particular witness is no longer useful. Fortunately, a user can efficiently prove in zero knowledge that he owns a membership/non-membership witness for a given value. This enables users to prove that they possess a witness for a given value without revealing the witness. These zero-knowledge proofs are extremely useful when implementing anonymous blacklisting systems.

As we discussed in section 1 in an anonymous blacklisting (AB) system we need servers to be able to prevent misbehaving users from accessing the server, without letting the server find the real identity of the users. The idea, therefore, is to have a system where we can punish users who do not obey the rules, but we do not want anyone to be able to link between users' activities.

The following is a list of properties an AB schemes can have and is borrowed from the work of Tsang et. al. [9]:

- **Misauthentication Resistance**: No unregistered user should be able to authenticate.
- **Revocablity**: Users blacklisted should not be able to authenticate successfully.

- **Anonymity**: Service provider should not be able to identify authenticating users.
- **Unlikablity**: Users authenticated connections cannot be linked.
- **Backward Unlikability**: Upon revocation, all the user's past authentications should remain anonymous.
- **Revocation Auditability**: Users should be able to check their revocation status before performing any action.
- ...

Each AB system may implement a subset of the above properties or may add some other properties. However, as we noted earlier, we are interested in AB systems which in addition to the above mentioned properties, have the following additional property:

- **Identity-escrow freeness** (particular to PEREA): There should exist no TTP that can infer the identity or pseudonyms of users.

In particular, we are discussing PEREA [9] system, which uses universal dynamic accumulators as discussed earlier. Here we briefly describe how it uses accumulators to achieve its goal of TTP-freeness. Figure 6 depicts the scheme and how the user and the server interact. Each user needs to authenticate to the service provider (SP) using tokens. Each token is issued and signed by the server in a previous session. The server can wait until client's action is performed and then decide to either issue the next token or not. Unfortunately, this cannot happen in most of the real-time applications. For example in the Wikipedia example discussed earlier, misbehaviours are usually reported after a while since moderators review changes periodically. Therefore, we need a system in which we can validate previous tokens of a user at each authentication. In PEREA, each user maintains a queue of his previous tokens and prove that all the elements in the queue are not blacklisted. Also, the server stores misbehaving tokens in a universal dynamic accumulator. This allows a user to provide a non-membership witness and thus prove that his token is not blacklisted, i.e. not accumulated in the accumulator. These non-membership witnesses are also stored on the client's side as seen in Figure 6 (values $w_0, ..., w_{k-1}$). Once a user needs to authenticate to the server, the following steps are taken:

1. The user obtains the SP's Blacklist and updates list of witnesses $(w_0, .., w_{K-1})$ for the tokens $(t_0, .., t_{K-1})$ in the queue. This step is necessary because the SP's blacklist might have changed from the previous authentication of the client, and so the list of witnesses should be updated accordingly. (Also note that this step allows for revocation auditability property discussed earlier, because the user can independantly figure out if any of his tokens is blacklisted before performing any action.)
2. The client generates token t_{K+1} for use in the next authentication and constructs the new queue $(t_1, .., t_{K+1})$.
3. The client also generates a commitment C of the new queue, and sends C and t_K to the SP.

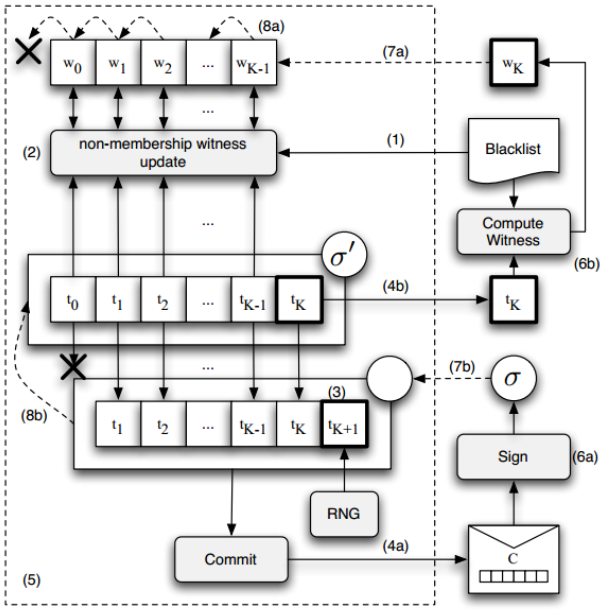

Fig. 6: PEREA Anonymous Blacklisting Scheme

4. User proves in zero-knowledge that he possesses non-membership witness for tokens $(t_0, .., t_{K-1})$.
5. User proves the integrity of the queue in zero-knowledge. This means that the user proves in zero knowledge that the new queue is obtained from the previous queue by removing token t_0 and adding token t_{K+1} (the user has the SP's signature on a commitment to the previous queue).
6. If the proofs in previous steps succeed, the SP will generate a new signature for C, and computes a witness w_K and sends them to the client.
7. User refreshes his list of witnesses and stores the new queue for subsequent authentication.

7 Conclusion and Future Work

We talked about the followings:

- One-way accumulators and how they are constructed.
- Secure dynamic accumulators and universal accumulators and how the additional properties added to them are useful for anonymous blacklisting schemes.
- PEREA: An anonymous blacklisting scheme employing universal dynamic accumulators for TTP-free blacklisting

Here is a list of future work we are interesed in:

– One way of extending PEREA is to use an accumulator to store the tokens used for contributing to an article or a project and give users credit according to their level of cooperation. We would like to see how we can implement this securely. This will greatly defeat sybil attacks where a user generates a large set of identities and attacks the system using them. By having a credit system, we can allow only specific users to perform certain tasks.
– Figuring out the window of misbehaving penalty and how many tokens should be given to each user, i.e., the number K in figure 6 seems like a possible future work.

References

1. Baric, N., and Pfitzmann, B. Collision-free accumulators and fail-stop signature schemes without trees. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques EUROCRYPT'97* (1997), 480–494.
2. Benaloh, J., and de Mare, M. One-way accumulators: a decentralized alternative to digital signatures. In *Workshop on the theory and application of cryptographic techniques on Advances in cryptology, EUROCRYPT '93* (1994), 274–285.
3. Camenisch, J., and Lysyanskaya, A. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '02* (2002), 61–76.
4. Cramer, R., and Shoup, V. Signature schemes based on the strong rsa assumption. *ACM Trans. Inf. Syst. Secur.* (2000), 161–185.
5. Henry, R., and Goldberg, I. Formalizing anonymous blacklisting systems. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy* (2011), 81–95.
6. Li, J., Li, N., and Xue, R. Universal accumulators with efficient nonmembership proofs. In *Applied Cryptography and Network Security*. 2007, 253–269.
7. Shamir, A. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. Comput. Syst.* (1983), 38–44.
8. Stinson, D. R. *Cryptography: Theory and Practice, Third Edition*. Chapman & Hall/CRC, 2006.
9. Tsang, P. P., Au, M. H., Kapadia, A., and Smith, S. W. Perea: towards practical ttp-free revocation in anonymous authentication. In *Proceedings of the 15th ACM conference on Computer and communications security, CCS '08* (2008), 333–344.