Anonymous Authentication with Revocation and Encryption Scheme for V2X

Taha Adeel Mohammed

Indian Institute of Technology Hyderabad

December 23, 2022

Reference

Title

PEREA: Towards Practical TTP-Free Revocation in Anonymous Authentication

Authors

- Patrick P. Tsang - Department of CS, Dartmouth College, USA
- Man Ho Au - University of Wollongong, Australia
- Apu Kapadia - Department of CS, Dartmouth College, USA
- Sean W. Smith - Department of CS, Dartmouth College, USA

Year of Publication

- 2008

Contents

Additional Design Goals

Additonal Design Goals

Revocation

- Users should be able to revoke their credentials if they wish to do so, hence allowing for self-revocation.
- The Road Side Units (RSUs)/Issuing Authority (IA) should be able to revoke the credentials of a user in a privacy preserving manner if it suspects that the user is malicious.
- The revocation should work with our randomized credentials and users should remain unlinkable accross multiple authentications.

High Level Overview

High Level Overview

- Similarly to our original scheme, the user has secrets $\alpha, \beta \in \mathbb{Z}_q^*$, which it uses to register with the IA, getting credentials (a, b, c, d).
- These credentials can be used for authentication and deriving the secret key for encrypted CAMs during authentication.
- Additionally, to support revocation of misbehaving users, we have each user also present a fresh ticket t_k to the RSU during authentication, and maintain a queue of the past K tickets it used for authentication.
- And we have the RSU maintain a blacklist of tickets belonging to misbehaving users.

High Level Overview

- Similarly to our original scheme, the user has secrets $\alpha, \beta \in \mathbb{Z}_q^*$, which it uses to register with the IA, getting credentials (a, b, c, d).
- These credentials can be used for authentication and deriving the secret key for encrypted CAMs during authentication.
- Additionally, to support revocation of misbehaving users, we have each user also present a fresh ticket t_k to the RSU during authentication, and maintain a queue of the past K tickets it used for authentication.
- And we have the RSU maintain a blacklist of tickets belonging to misbehaving users.

High Level Overview

- During authentication, we have the user prove in zero-knowledge that the last K tickets in its queue haven't been blacklisted by the IA.
- We also have the user prove in zero knowledge that the queue of last K tickets is correctly formed.
- After verifying the user's credentials, the integrity of the queue, and that the user hasnt been blacklisted, the RSU computes a witness for the ticket t_k and a signature on the queue and shares it with the user.
- The user can then use this signature and witness during its next authentication to prove to the RSU in zero-knowledge of the validity of his credentials.

High Level Overview

- During authentication, we have the user prove in zero-knowledge that the last K tickets in its queue haven't been blacklisted by the IA.
- We also have the user prove in zero knowledge that the queue of last K tickets is correctly formed.
- After verifying the user's credentials, the integrity of the queue, and that the user hasnt been blacklisted, the RSU computes a witness for the ticket t_k and a signature on the queue and shares it with the user.
- The user can then use this signature and witness during its next authentication to prove to the RSU in zero-knowledge of the validity of his credentials.

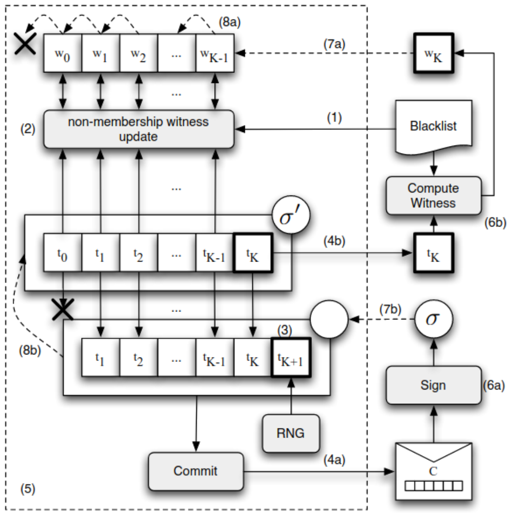
High Level Overview

- During authentication, we have the user prove in zero-knowledge that the last K tickets in its queue haven't been blacklisted by the IA.
- We also have the user prove in zero knowledge that the queue of last K tickets is correctly formed.
- After verifying the user's credentials, the integrity of the queue, and that the user hasnt been blacklisted, the RSU computes a witness for the ticket t_k and a signature on the queue and shares it with the user.
- The user can then use this signature and witness during its next authentication to prove to the RSU in zero-knowledge of the validity of his credentials.

High Level Overview

- During authentication, we have the user prove in zero-knowledge that the last K tickets in its queue haven't been blacklisted by the IA.
- We also have the user prove in zero knowledge that the queue of last K tickets is correctly formed.
- After verifying the user's credentials, the integrity of the queue, and that the user hasnt been blacklisted, the RSU computes a witness for the ticket t_k and a signature on the queue and shares it with the user.
- The user can then use this signature and witness during its next authentication to prove to the RSU in zero-knowledge of the validity of his credentials.

High Level Overview



Building Blocks

Preliminaries

Bilinear Pairings

Let G and G_T be groups of prime order q. A map $e : G \times G \rightarrow G_T$ must satisfy the following properties:

- *Bilinearity*: a map $e : G \times G \rightarrow G_T$ is bilinear if $e(a^x, b^y)t = e(a, b)^{xy}$;
- *Non-degeneracy:* for all generators $g, h \in G$, $e(g, h)$ generates G_T ;
- Efficiency: There exists an efficient algorithm $BMGen(1^k)$ that outputs (q, G, GT, e, g) to generate the bilinear map and an efficient algorithm to compute $e(a, b)$ for any $a, b \in G$.

Hardness Assumptions

The security of our scheme depends on the following number-theoretic hardness assumptions:

Strong RSA Assumption

The Strong RSA Assumption says that there exists no PPT algorithm which, on input N and $u \in \mathbb{Z}_N^*$, returns $e > 1$ and v such that $v^e = u \mod N$, with non-negligible probability (in λ).

DDH Assumption

The Decisional Diffie-Hellman (DDH) Assumption says that there exists no PPT algorithm which, on input quadruple $(g, g^a, g^b, g^c) \in QR_N^4$,, where $a, b \in_R \mathbb{Z}_{|QR_N|}$, and $c \in_R Z_{|QR_N|}$ or $c = ab$ with equal probability, correctly distinguishes which is the case with probability non-negligibly (in λ) greater than $1/2$.

ZKPoK Protocols

- A *Zero-Knowledge Proof-of-Knowledge (ZKPoK)* protocol is a protocol in which a prover convinces a verifier that some statement is true without the verifier learning anything except the truth of the statement.
- Eg: $PK\{(x) : y = g^x\}$ denotes a ZKPoK protocol that proves the knowledge of an integer x such that $y = g^x$ holds, where the value x is secret to the prover.

Tickets and Queues

- The user picks a ticket uniformally at random from the set of primes of atmost l_t bits, where l_t is 166.
- As there are atleast 2^{160} tickets, the probability of two randomly chosen tickets colliding is atmost 2^{-80}.
- We represent the queue maintained by the user as

$$Q = \{t_0, t_1, \ldots, t_{k-1}\}$$

Tickets and Queues

- The user picks a ticket uniformally at random from the set of primes of atmost l_t bits, where l_t is 166.
- As there are atleast 2^{160} tickets, the probability of two randomly chosen tickets colliding is atmost 2^{-80}.
- We represent the queue maintained by the user as

$$Q = \{t_0, t_1, \ldots, t_{k-1}\}$$

Tickets and Queues

- The user picks a ticket uniformally at random from the set of primes of atmost l_t bits, where l_t is 166.
- As there are atleast 2^{160} tickets, the probability of two randomly chosen tickets colliding is atmost 2^{-80}.
- We represent the queue maintained by the user as

$$Q = \{t_0, t_1, \ldots, t_{k-1}\}$$

Accumulator Scheme for Tickets

- We require that the users should be able to successfully prove that they have not been blacklisted by the IA, without revealing their tickets.
- Hence for this, we make use of Universal Dynamic Accumulators (UDAs), introduced by Li et al, which allows for an efficient ZKPoK of non-membership in time independant of number of accumulated values.

Accumulator Scheme for Tickets

- We require that the users should be able to successfully prove that they have not been blacklisted by the IA, without revealing their tickets.
- Hence for this, we make use of Universal Dynamic Accumulators (UDAs), introduced by Li et al, which allows for an efficient ZKPoK of non-membership in time independant of number of accumulated values.

Accumulator Scheme for Tickets

The accumulator has a value V when values $S_T = \{t_0, t_1, \ldots, t_T\}$ are accumulated in it. We can perform the following operations with the help of the accumalator:

- $\mathit{Accumulate}(V, t_k)$ which outputs the updated value of the accumulator V' that now also contains t_k.
- Each ticket t not in S_T has a corresponding non-membership witness w, such that $\mathit{IsNonMember}(t, V, w) = 1$ holds.
- This w can be computed using knowledge of the secret key of the accumulator using $\mathit{ComputeWitness}(t, V, sk_{acc})$.
- Given (t, w) for an accumalator value V, anyone can update the witness w to a new witness w' for the updated accumulator value V' using $\mathit{UpdateWitness}(w, t, V, V')$.

Accumulator Scheme for Tickets

The accumulator has a value V when values $S_T = \{t_0, t_1, \dots, t_T\}$ are accumulated in it. We can perform the following operations with the help of the accumalator:

- $Accumulate(V, t_k)$ which outputs the updated value of the accumulator V' that now also contains t_k.
- Each ticket t not in S_T has a corresponding non-membership witness w, such that $IsNonMember(t, V, w) = 1$ holds.
- This w can be computed using knowledge of the secret key of the accumulator using $ComputeWitness(t, V, sk_{acc})$.
- Given (t, w) for an accumalator value V, anyone can update the witness w to a new witness w' for the updated accumulator value V' using $UpdateWitness(w, t, V, V')$.

Accumulator Scheme for Tickets

The accumulator has a value V when values $S_T = \{t_0, t_1, \ldots, t_T\}$ are accumulated in it. We can perform the following operations with the help of the accumalator:

- $Accumulate(V, t_k)$ which outputs the updated value of the accumulator V' that now also contains t_k.
- Each ticket t not in S_T has a corresponding non-membership witness w, such that $IsNonMember(t, V, w) = 1$ holds.
- This w can be computed using knowledge of the secret key of the accumulator using $ComputeWitness(t, V, sk_{acc})$.
- Given (t, w) for an accumalator value V, anyone can update the witness w to a new witness w' for the updated accumulator value V' using $UpdateWitness(w, t, V, V')$.

Accumulator Scheme for Tickets

The accumulator has a value V when values $S_T = \{t_0, t_1, \ldots, t_T\}$ are accumulated in it. We can perform the following operations with the help of the accumalator:

- $Accumulate(V, t_k)$ which outputs the updated value of the accumulator V' that now also contains t_k.
- Each ticket t not in S_T has a corresponding non-membership witness w, such that $IsNonMember(t, V, w) = 1$ holds.
- This w can be computed using knowledge of the secret key of the accumulator using $ComputeWitness(t, V, sk_{acc})$.
- Given (t, w) for an accumalator value V, anyone can update the witness w to a new witness w' for the updated accumulator value V' using $UpdateWitness(w, t, V, V')$.

Protocol for Queue Signing

- The RSU needs to verify the integrity of the queue, since otherwise a user could circumvent revocation by fabricating a queue with an incorrect set of K tickets.
- For this, normal digital signatures are not sufficient, as they will immediately allow the RSU to link the user to past actions.
- We require the verification to happen without revealing either the queue or the signature on it.
- For this, we utilize the signature scheme given by Camenisch and Lysyanskaya for a block of messages, adapted for our scheme.

Protocol for Queue Signing

- The RSU needs to verify the integrity of the queue, since otherwise a user could circumvent revocation by fabricating a queue with an incorrect set of K tickets.
- For this, normal digital signatures are not sufficient, as they will immediately allow the RSU to link the user to past actions.
- We require the verification to happen without revealing either the queue or the signature on it.
- For this, we utilize the signature scheme given by Camenisch and Lysyanskaya for a block of messages, adapted for our scheme.

Protocol for Queue Signing

- The RSU needs to verify the integrity of the queue, since otherwise a user could circumvent revocation by fabricating a queue with an incorrect set of K tickets.
- For this, normal digital signatures are not sufficient, as they will immediately allow the RSU to link the user to past actions.
- We require the verification to happen without revealing either the queue or the signature on it.
- For this, we utilize the signature scheme given by Camenisch and Lysyanskaya for a block of messages, adapted for our scheme.

Protocol for Queue Signing

- The RSU needs to verify the integrity of the queue, since otherwise a user could circumvent revocation by fabricating a queue with an incorrect set of K tickets.
- For this, normal digital signatures are not sufficient, as they will immediately allow the RSU to link the user to past actions.
- We require the verification to happen without revealing either the queue or the signature on it.
- For this, we utilize the signature scheme given by Camenisch and Lysyanskaya for a block of messages, adapted for our scheme.

Protocol for Queue Signing

This signature scheme allows us to perform the following necessary operations and ZKPoKs:

- **Proof of knowledge of a signed queue:** This protocol allows a user to prove to the RSU the possession of a valid signature without revealing the queue and signatures themselves.
- **Proof of relation between two queues:**
  - During authentication, the user updates his queue from Q to $Q' = Q.Enq(t_k).Deq()$ for use during the next authentication.
  - This proof allows the user to convince to the RSU that Q' was indeed correctly updated from Q.

Protocol for Queue Signing

This signature scheme allows us to perform the following necessary operations and ZKPoKs:

- **Proof of knowledge of a signed queue:** This protocol allows a user to prove to the RSU the possession of a valid signature without revealing the queue and signatures themselves.
- **Proof of relation between two queues:**
  - During authentication, the user updates his queue from Q to $Q' = Q.Enq(t_k).Deq()$ for use during the next authentication.
  - This proof allows the user to convince to the RSU that Q' was indeed correctly updated from Q.

Protocol for Queue Signing

This signature scheme allows us to perform the following necessary operations and ZKPoKs:

- **Proof of knowledge of a signed queue:** This protocol allows a user to prove to the RSU the possession of a valid signature without revealing the queue and signatures themselves.
- **Proof of relation between two queues:**
  - During authentication, the user updates his queue from Q to $Q' = Q.Enq(t_k).Deq()$ for use during the next authentication.
  - This proof allows the user to convince to the RSU that Q' was indeed correctly updated from Q.

Protocol for Queue Signing

This signature scheme allows us to perform the following necessary operations and ZKPoKs:

- **Proof of knowledge of a signed queue:** This protocol allows a user to prove to the RSU the possession of a valid signature without revealing the queue and signatures themselves.
- **Proof of relation between two queues:**
  - During authentication, the user updates his queue from Q to $Q' = Q.Enq(t_k).Deq()$ for use during the next authentication.
  - This proof allows the user to convince to the RSU that Q' was indeed correctly updated from Q.

Modified Scheme Construction

Modified Scheme Construction

Using the above mentioned building blocks, we can construct a modified scheme as follows:

Setup

- For the randomizable signatures using bilinear pairings, the IA chooses a Type-3 pairing with the parameters $(e, g_1, g_2, g, G_1, G_2, G)$, and $(sk, pk) = ((x, y), (X = g_2^x, Y = g_1^y))$, where $x, y \in \mathbb{Z}_q^*$
- Depending on the system requirements, the IA decides on an appropriate revokation window size K. For our scheme, small values of K, such as $K = 10$, would also be sufficient.
- The IA also initializes the accumalator, blacklist, and the parameters of the signature scheme for the queue.

Modified Scheme Construction

Registration

- The user has secrets $(\alpha, \beta) \in_R \mathbb{Z}_q^*$, and sends $req = (a = g_1^{\beta}, b = a^{\alpha})$ to the IA. The IA verifies the user and outputs the signature $\sigma = (a, b, c = a^x, d = (bc)^y)$ to the user.
- Also the user picks a random ticket t and initializes his queue as $Q = \{\hat{t}, \hat{t}, \ldots, \hat{t}, t\}$, where $\hat{t}$ is a dummy ticket provided by the IA.
- Then the user engages in the ZKPoKs to prove his queue is well formed, and to get the witness for the ticket and signature for the queue.
- Finally, the user stores his credentials as

$$cred = (\sigma, t, Q, (w_i)_{i=0}^{K-1}, \sigma', BL, V)$$

Modified Scheme Construction

Authentication

- **Randomizable Signature check:** The user shares $\sigma^r = (a^r, b^r, c^r, d^r)$, which the RSU verifies by checking that $e(a^r, X) = e(c^r, g_2)$ and $e(d^r, g_2) = e(bc^r, Y)$.
- **Blacklist Check:** The user obtains the current blacklist from the RSU, and verifies that he hasn't been revoked.
- **Request for authentication:**
  - Using the updated value of the blacklist, the user updates the witness for each ticket in his queue, and generates a fresh ticket t^*.
  - Then he engages with the RSU in the ZKPoKs to prove that his queue is well formed, and that none of his tickets have been revoked.
- **Refreshment issuing:** After verifying the user, the RSU computes the witness and signature for the queue, and sends it to the user.
- **Credential Refreshment:** Finally the user updates his credentials for the next authentication using the signature and witness.

Modified Scheme Construction

Revocation

- To blacklist a misbehaving user who provided t_k, the RSU/IA updates the blacklist as $BL' = BL \cup \{t_k\}$, and updates the accumulator as $V' = Accumulate(V, t_k)$.
- The next time the misbehaving user tries to authenticate, he will be unable to prove that t_k from his queue is not in the accumulator. Hence he won't be authenticated.

Key generation and Encryption Scheme

After the authentication in the above steps, the rest of the construction for key generation and sharing of encrypted CAMs can be done similarly to the original scheme.

Limitations

Limitations

- While the above proposed scheme allows us to have privacy enhanced revocation in V2X, it significantly impacts the efficiency of the scheme.
- For each authentication, the user now has to perform the ZKPoKs for each value in the queue. This significantly increases the computation and communication overheads, which is not very acceptable for V2X applications.

Thank you!