Anonymous Authentication with Revocation and Encryption Scheme for V2X

Introduction

Below we describe a scheme that allows registered users to authenticate themselves with an RSU, and once successfully authenticated, derive a symmetric key with the help of the RSU and communicate with its adjacent peers using CAMs encrypted using that key. ~~We also~~

We also describe a way in which the RSU/IA can revoke the authentication of a user in a privacy preserving manner if it suspects that the user is malicious

High-level Overview

The user has secrets α, β in $\mathbb{Z}_q^*$, which it uses to register with the IA, getting credentials (a, b, c, d), which can be used for authentication and deriving the secret key for CAMs during authentication.

Additionally, to prove that the user is not blacklisted, the user generates a fresh ticket t_K for each authentication, and maintains a queue of the past K (revokation window size) authentication tickets $Q = \{t_0, t_1, \ldots, t_{K-1}\}$, while the verifier maintains a blacklist of tickets belonging to misbehaving users. During authentication, the user shares t_K and proves in zero-knowledge that the last k tickets in its queue haven't been blacklisted by the IA. After verifying the users credentials and the integrity of the queue of last k tickets, the RSU computes a witness for t_K, which can be used by the user in its next authentication to prove in zero knowledge the validity of ticket t_K.

Building Blocks

I) Preliminaries

→ Notation :-

* $a \xleftarrow{R} X$: Denotes that a is chosen uniformly at random from set X
* $\mathbb{Z}_q$: Set of integers modulo q
* Λ_ℓ : Set of integers of size atmost ℓ-bits
* Π_ℓ : Set of primes of size atmost ℓ-bits
* $\Delta(\ell, \delta)$: The set $\{2^{\ell-1}, \ldots, 2^{\ell-1} + 2^{\delta} - 1\}$
* Safe Prime: p s.t. p and $\frac{p-1}{2}$ are both prime
* ℓ-bit safe prime product : Product of two $\lfloor \frac{\ell}{2} \rfloor$-bit safe primes.
* QR_N : Set of quadratic residues modulo N.
* $\phi(N)$: Euler's totient of N

→ Bilinear Pairings

→ Hardness Assumptions

i) Co-computational Diffie-Hellman (Co-CDH) assumption.

ii) Decisional Bilinear Diffie-Hellman (DBDH) assumption

iii) Decisional Diffie-Hellman assumption

iv) Strong RSA assumption

→ ZKPoK Protocols

* The notation introduced by Camenisch and Stadler is used in our our scheme to represent the ZKPoK Protocols used in our scheme.
* For example, $PK\{(x): y = g^x\}$ denotes a ZKPoK protocol that proves knowledge of an integer x s.t. $y = g^x$ holds.

→ Randomizable Signatures (To generate user credentials)

II) Tickets and Queues

* The user picks a ticket uniformally at random from the set Π_{ℓ_t}, where $\ell_t = 166$. i.e.

$$t \xleftarrow{R} \Pi_{\ell_t}$$

* Note that $|\Pi_{\ell_t}| \geq 2^{160}$. Hence the probabilty of two randomly chosen tickets colliding is atmost 2^{-80} (By the birthday paradox)
* We set the ticket domain to be

$$\mathcal{T} = \{-2^{\ell_T}+1, \ldots, 2^{\ell_T}-1\}, \text{ where } \ell_T = 330$$

* A queue of size k is a sequence of k tickets, supporting the enqueuing (Enq) and dequing (Deq) operation.
* $Q[i]$ denotes the i-th least recently ~~ticket~~ enqueued ticket in the queue Q of size $(K+1)$, where K is the revocation window size.
* Note that the domain of $Q = \mathcal{T}^{K+1} = \mathcal{Q}$

III) Accumulator Scheme for tickets

(To Prove user is not revoked)

* We ~~use~~ make use of Universal Dynamic Accumulators (UDAs) introduced by Li et al, which allows for an efficient zero-knowledge proof of non-membership, in time independent of number of accumulated values. The IA blacklists users by accumulating their tickets into UDAs.
* Below we describe a construction of UDAs adapted for our notation and requirements. We call the modified scheme TicketAcc.

→ Key generation:-

* Input security parameter: $param_{acc} = l_N$ (= 1024 recommended)
* Pick $N = pq$, ~~an~~ l_N-bit safe prime product and $g \xleftarrow{R} QR_N$
* Output the accumulator's private and public key resp.

$$sk_{acc} = \phi(N)$$

$$pk_{acc} = (l_N, N, g)$$

→ Accumulating tickets

* Input: ticket $t \in T$ and accumulator value V.

$$Accumulate(V, t) \longrightarrow V' = V^t \pmod N$$

* Similarly, for set $S_T = \{t_1, t_2, \dots, t_L\}$,

$$Accumulate(V, S_T) \longrightarrow V' = V^{t_1 t_2 \dots t_L} \pmod N$$

* An accumulator value ~~initially~~ is initially g. Hence we define

$$Accumulate(S_T) = Accumulate(g, S_T) = g^{t_1 t_2 \dots t_L} \pmod N$$

→ <u>Non-membership witnesses</u>

* If $V = \text{Accumulate}(S_T)$ for some $S_T \subset T$ and $t \in T \setminus S_T$ then $\exists$ a non-membership witness

$$w = (a, d) \in \mathbb{Z}_{[\frac{n}{4}]} \times QR_N$$

for t w.r.t V such that $1 = \text{IsNonMember}(t, V, w)$, where

$$\text{IsNonMember}(t, V, w) = \begin{cases} 1 & \text{if } V^a \equiv d^t g \\ 0 & \text{otherwise} \end{cases}$$

* A valid prover can convince a verifier that t was not accumulated in V using w, without revealing t or w, by conducting

$$PK\{(t, w) : 1 = \text{IsNonMember}(t, V, w)\}$$

The construction and security of above protocol has been given by Li et al. and it runs with a time complexity of $O(1)$, i.e. independent of number of accumulated values.

* A witness $w = (a, d)$ for t w.r.t V can be computed using knowledge of sk_{acc} as:

$$\text{ComputeWitness}(t, V, sk_{acc}) \longrightarrow w$$

Implementation of above function can be reffered to in Li et al.'s paper.

→ Update of non-membership Witnesses

* Given witness w s.t. $IsNonMember(t, V, w) = 1$, when V gets updated to V' via the accumulation of a new ticket $t' \in T \setminus \{t\}$ into it, one can compute, without knowledge of sk_{acc}, an updated witness w' s.t. $IsNonMember(t, V', w') = 1$, as

$$UpdateWitness(w, t, V, t') \rightarrow w'$$

* Again, refer to Li et al.'s scheme for implementation.
* Similarly for $S_T \subset T \setminus \{t\}$, we define

$$UpdateWitness(w, t, V, S_T) \rightarrow w'$$

as repeatedly updating the witness for each ~~on~~ ticket $\in S_T$.

* Complexity of above operation is linear in size of S_T, i.e. the number of new values accumulated.

IV) Protocol for queue signing

(To prove integrity of the queue)

* 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, the RSU signs the queue on a successful authentication, so that it can be convinced of its integrity during the next authentication.
* This must be done without revealing the queue on the signature, as otherwise the user's actions can be linked. Hence, for the above purpose, we use the signature scheme given by Camenisch and Lysyanskaya, as shown below (Queue Sig):

→ **Key generation:-**

* Input security parameters: $param_{sig} = (l_N, l_s, l_e, l_T, l, \delta_r)$
* Choose $N = pq$, an l_N-bit safe prime, and $b, c, g_0, \ldots g_K \xleftarrow{R} QR_N$
* Output: $sk_{sig} = \phi(N)$
  $pk_{sig} = (param_{sig}, N, b, c, (g_i)_{i=0}^{K})$

→ **Request for signature:-**

* To request a signature on a commited queue $Q = (t_i)_{i=0}^{K} \in \mathcal{Q}$, Alice picks $r \xleftarrow{R} \Delta(l_N, \delta_r)$, commits Q:

$$\text{Commit}(Q, r) \rightarrow C = c^r \prod_{i=0}^{K} g_i^{t_i} \pmod N,$$

and then sends the commitment C to the RSU.

* Proof of correctness:- Alice (as the prover) then conducts the following protocol with the verifier to prove that the commitment was constructed correctly:

$$PK\{(Q, r) : C = \text{Commit}(Q, r) \wedge Q \in \mathcal{Q} \wedge r \in \Delta(l_N, \delta_r)\}$$

The RSU proceeds only if the protocol succeeds.

→ Signing

* The RSU signs and returns to Alice a signature $\tilde{\sigma}$ on C using its private key sk_{sig}:

$$Sign(C, sk_{sig}) \rightarrow \tilde{\sigma} = (r', e, v), \text{ where } r' \xleftarrow{R} \Lambda_{\ell_s},$$

$e \xleftarrow{R} \Pi_{\ell_e}$ and $v = (bc^{r'} C)^{e^{-1} \bmod \phi(N)} \pmod N$

→ Finalizing

* Alice finalizes the signature $\tilde{\sigma} = (r', e, v)$ on the commitment C into a signature σ on her queue Q:

$$Finalize(\tilde{\sigma}, r) \rightarrow \sigma = (r + r', e, v)$$

* She proceeds only if the signature verifies, i.e. $Verify(Q, \sigma) = 1$, where

$$Verify(Q, \sigma) = \begin{cases} 1 & \text{if } v^e \equiv bc^s \prod_{i=0}^{k} g_i^{t_i} \wedge e > 2^{\ell_e - 1} \\ 0 & \text{otherwise} \end{cases}$$

Proof of knowledge of a signed queue

* The ~~tt~~ below protocol allows a user to prove to the RSU the possession of a valid signature without revealing the queue and signature themselves.

$$PK\{(Q, \sigma) : 1 = Verify(Q, \sigma) \wedge Q \in \mathcal{Q}\}$$

The construction for above protocol is standard for CL-signature on blocks of messages, and can be reffered to.

Proof of relation between two queues

* During authentication, Alice updates her current queue from Q' to $Q = Q'.\mathrm{Enq}(t^*).\mathrm{Deq}()$, for use during the next authentication, where t^* is the new ticket.
* The below protocol convinces the verifier that Q was indeed correctly updated from Q'.

$$PK\{(Q', Q, t) : Q = Q'.\mathrm{Enq}(t^*).\mathrm{Deq}() \wedge Q' \in \mathcal{Q} \wedge t^* \in \mathcal{T}\}$$

This protocol can be constructed as follows:

- Compute the commitments C_0, C_1 using $r_0, r_1 \xleftarrow{R} \Delta(l_N, S_r)$ on Q', Q and conduct the following protocol:

$$PK\{(r_0, r_1, (t_i)_{i=0}^{K+1}) : \bigwedge_{l=0,1} C_l \equiv c_2 \prod_{i=0}^{K} g_i^{t_{i+l}}\}$$

Which can be done using standard protocols for proving relations among components of a discrete logarithm representation of a group of elements.

* The above ZKPoK Protocols and signature protocol have an $O(K)$ computational and communicational complexity between Alice and RSU.

Construction

Below we ~~list~~ ~~out~~ ~~the~~ describe the construction of our Privacy Preserving Authentication ~~and~~ with revo~~k~~cation and Encryption Scheme.

→ Setup

* For the randomizable signatures using bilinear pairings, for the randomizable signatures output the type-3 bilinear pairing
  - Given security parameter k, output the type-3 bilinear pairing parameters $(e, g_1, g_2, g, G_1, G_2, G) \xleftarrow{R} Gen(1^k)$
  - The IA generates $sk = (x, y) \xleftarrow{R} \mathbb{Z}_q^*$ as its secret keys and publishes $pk = (X = g_2^x, Y = g_2^y)$ as its public keys.
* The IA decides on an appropriate revokation window size K, and on ~~d~~ input parameters $param_{acc}$ & $param_{sig}$ generates (sk_{acc}, pk_{acc}) and (sk_{sig}, pk_{sig}) respectively, according to the scheme in previous section.
* The IA also picks a prime $\hat{t} \in \Pi_{\ell_e}$, which is used to fill a user's queue as the default value during registration.
* Initially, the IA's ~~Block~~ blacklist $BL = \phi$, accumulator $V = g$, and ticket-list $TL = \phi$
* The IA creates a private key $sk_{IA} = (sk, sk_{sig}, sk_{acc})$ and a public key $pk_{IA} = (pk, \hat{t}, pk_{sig}, pk_{acc})$

→ <u>Registration</u>

- The user has secrets $(\alpha, \beta) \xleftarrow{R} \mathbb{Z}_q^*$ and sends $req = (a = g_1^{\beta}, b =$ to the issuer. The issuer verifies the vehicle and uses its secret keys to compute $(c, d) = (a^x, (a^x \cdot c)^y)$ and outputs the signature $\sigma = (a, b, c, d)$

- i) Also, the user picks $t^* \xleftarrow{R} \Pi_{\ell_t}$ and initializes their queue Q' as $Q' = (\hat{t}, \hat{t}, \ldots, \hat{t}, t^*)$. Then the user sends $C = \text{Commit}(Q', r)$, where $r \xleftarrow{R} \Delta(\ell_N, \delta_r)$, to the IA.

ii) (Proof of correctness) Then the user engages in the following protocol with the IA.

$$PK \left\{ (Q', r) : \bigwedge_{i=0}^{K-1} \hat{t} = Q'[i] \wedge C = \text{Commit}(Q', r) \wedge Q' \in \mathcal{Q} \right\}$$

Above protocol is similar to the one in TicketAcc, & hence can be constructed similarly. The IA proceeds ~~after~~ only if this protocol terminates successfully.

iii) (Credential issuing): The IA computes $\tilde{\sigma} = \text{Sign}(C, sk_{sig})$ and $\hat{\omega} = \text{ComputeWitness}(\hat{t}, V', sk_{acc})$, where $V' = \text{Accumulate}(BL')$. The IA returns $(\tilde{\sigma}, \hat{\omega}, BL', V')$ to the user.

iv) (Credential finalizing): The user computes $\sigma' = \text{Finalize}(\tilde{\sigma}, r)$ and proceeds only if $V' = \text{Accumulate}(BL')$, $1 = \text{Verify}(Q', \sigma')$ and $1 = \text{IsNonMember}(\hat{t}, V', \hat{\omega})$

- The user then stores its credentials as

$$cred = (\sigma, \sigma', Q', (\omega_i)_{i=0}^{K-1}, BL', V'), \text{ where } \omega_i = \hat{\omega} \text{ for } i \leq K-1$$

→ Authentication

We now describe the authentication protocol between the user and the RSU.

i) Blacklist examination:

* The user obtains the current blacklist BL from the RSU, and after asserting that none of the tickets in Q' have been revoked, ~~she~~ the user continues.
* Also, the user finds $\Delta_{BL} = BL \setminus BL'$, i.e. set of newly blacklisted tickets.

ii) Request for authentication:

* The user generates a new ticket $t^* \leftarrow \Pi_{\ell_t}$ and $r \xleftarrow{\$} \Delta(\ell_N, \delta_r)$, and computes:

$$
\begin{aligned}
t_K &= Q'[K] \\
Q &= Q'.\mathrm{Enq}(t^*).\mathrm{Deq}() \\
C &= \mathrm{Commit}(Q, r) \\
V &= \mathrm{Accumulate}(V', \Delta_{BL}), \\
w_i &= \mathrm{WitnessUpdate}(w_i', Q'[i], V, \Delta_{BL}), \text{ for } i \in [0, K)
\end{aligned}
$$

* The user then sends (t_K, C) to the RSU, and after the RSU verifies that t_K is a fresh prime in Π_{ℓ_t}, it adds t_K to TL.
* Proof of correctness: The user then engages in the below ZKPoK protocol to convince the RSU that he hasn't been revoked yet, t_K is a well-formed ticket and C is a well formed commitment of the user's next queue.

$$
\begin{aligned}
PK\{(Q', \sigma', (w_i)_{i=0}^{K-1}, t^*, Q, r) :\ & \\
t_K &= Q'[K] \ \wedge \\
1 &= \mathrm{Verify}(Q', \sigma') \ \wedge \\
\textstyle\bigwedge_{i=0}^{K-1} 1 &= \mathrm{IsNonMember}(Q'[i], V, w_i) \ \wedge \\
Q &= Q'.\mathrm{Enq}(t^*).\mathrm{Deq}() \ \wedge \\
C &= \mathrm{Commit}(Q', r) \ \wedge \\
Q' &\in \mathcal{Q} \wedge t^* \in \mathcal{T} \wedge r \in \mathcal{R}\}
\end{aligned}
$$

Above protocol can be built using the individual protocols in previous section

iii) Refreshment issuing:

* RSU then computes and shares the following with the user:

$$\tilde{\sigma} = \mathrm{Sign}(Q, sk_{sig}), \text{ and}$$

$$w_k = \mathrm{ComputeWitness}(V, t_K, sk_{acc})$$

iv) Credential refreshment:

* The user finalizes the signature σ', i.e

$$\sigma' = \mathrm{Finalize}(\tilde{\sigma}, Q, r)$$

and checks for correctness:

$$1 \stackrel{?}{=} \mathrm{Verify}(Q, \sigma')$$

$$1 \stackrel{?}{=} \mathrm{IsNonMember}(t_K, V, w_K)$$

* The user updates credentials for next authentication as

$$(Q', \sigma') = (Q, \sigma')$$

$$(w_0', w_1', \ldots w_{K-2}') = (w_1, \ldots, w_{K-1})$$

$$w_{K-1}' = w_K$$

$$(BL', V') = (BL, V)$$

v) Randomizable Signature verification:

* The user shares (a^r, b^r, c^r, d^r), which the RSU verifies by checking $e(a^r, X) \stackrel{?}{=} e(c^r, g_2)$ and $e(d^r, g_2) \stackrel{?}{=} e(bc, Y)$
* After above verification succeeds and the user proves he is not blacklisted, the RSU follows the scheme for Key Generation Mechanism, following which a valid user can successfully encrypt and decrypt their CAMs. efficiently

→ Revocation

* To blacklist the user who provided t_K, the ~~SP~~ RSU updates its blacklist as $BL = BL \cup \{t_K\}$ and the corresponding accumulated value as

$$V = \mathrm{Accumulate}(V', t_K^*)$$