Mini-Project Report: Lightweight privacy preserving authentication and Encryption scheme in V2X with Self-Revocation

Prashanth Sriram S - CS20BTECH11039

December 2022

1 Authentication Scheme

- **Setup** $\mathcal{S}(1^k)$**:** With the security parameter k, output the type-3 bilinear pairing parameters $(q, e, g_1, g_2, \mathbf{g}, G_1, G_2, \mathbf{G}) \xleftarrow{R} \mathcal{G}en(1^k)$
  Output the PKE scheme: $(GenPKE, Enc, Dec) \leftarrow GenPKEScheme(1^k)$
- **Keygen** $\mathcal{K}(1^k)$**:**
  1. The enrollment authority generates $sk = (x_0, y) \xleftarrow{R} \mathbb{Z}_q^*$ as its secret keys and publishes $pk = (X_0 = g_2^{x_0}, Y = g_2^y)$ as its public keys
  2. The user generates $(\alpha, \beta) \xleftarrow{R} \mathbb{Z}_q^*$ as its secrets. User also generates $(sk_{user}, pk_{user}) \leftarrow GenPKE(1^k)$. User sends $req = (a = g_1^\beta, b = a^\alpha, pk_{user})$ to the issuer for obtaining the credential
- **InitIA** $\mathcal{I}(1^k)$**:** The issuing authority creates an empty table T with $A, B, pk, C_{enc}, D_{enc}$ as its columns. This Table will contain the entries for each of the valid users. (The issuing authority publishes the pk, C_{enc}, D_{enc} part of T before every epoch begins)
- **GenCred** $\mathcal{G}(sk, req, T)$, where $sk = (x, y)$ is the secret key of the issuing authority this epoch:
  1. The issuer verifies the vehicle and uses its secret keys (x, y) to compute $c = a^x, d = (a^\alpha.c)^y = a^{y(\alpha+x)}$ and then outputs the signature $\sigma = (a, b, c, d)$ to the user.
  2. It computes $C_{enc} = Enc(pk_{user}, c)$ and $D_{enc} = Enc(pk_{user}, d)$
  3. It then adds $(a, b, pk_{user}, C_{enc}, D_{end})$ to the table T.

- **IssuerUpdate** $\mathcal{U}(1^k, sk, T)$:, run by the issuing authority before the next epoch begins,

  1. Issuing authority picks a new $x' \xleftarrow{R} \mathbb{Z}_q^*$ and publishes $X^{'} = g_2^{x^{'}}$, thus making the secret and public keys for the next epoch as $sk' = (x', y)$ and $pk' = (X', Y)$ and publishes the new public keys.
  2. For each entry in T:
     (a) Let the entry be $(A_i, B_i, pk_{user_i}, C_{enc_i}, D_{enc_i})$
     (b) The issuing authority computes new $c = A_i^x, d = (B.c)^y$
     (c) The I.A. updates $C_{enc_i} = Enc(pk_{user_i}, c)$ and $D_{enc_i} = Enc(pk_{user_i}, d)$ in T
  3. The issuing authority publishes the $pk_{user_i}, C_{enc_i}, D_{enc_i}$ columns of T online, let this be called T_{pub}

  (Note, the I.A. only needs to go online after completing step 2 and publish the updates all at once.)

- **UserUpdate** $(sk_{user}, pk_{user}, pk', T_{pub})$**:**, run by each user to update their credentials for the next epoch, (run after the issuer has updated T_{pub})

  1. The user goes online and searches for the entry with their pk_{user} in T_{pub} and download that entry - $(pk_{user}, C_{enc}, D_{enc})$
  2. User uses their secret key to decrypt to get the updated credentials, $c' = Dec(sk_{user}, C_{enc})$, $d' = Dec(sk_{user}, D_{enc})$. User now uses $\sigma' = (a, b, c', d')$ as their credentials for the next epoch.
  3. User runs $\mathcal{V}(\sigma', pk')$ to verify whether the new credentials are correct.

  (Note: In case there were multiple users with the same public key, the user needs to download all such entries in T_{pub} and try to compute σ' and run $\mathcal{V}$ to verify if this is the correct entry for their credentials. But, this is unlikely and also there are no security leaks here, since just knowing the credentials is not enough to derive the symmetric key from the RSU)

- **Verify** $\mathcal{V}(\sigma, pk)$ **:** Uses the public keys of the issuer this epoch and the credential of the user as input and verifies whether $e(a, X) \stackrel{?}{=} e(c, g_2)$ and $e(d, g_2) \stackrel{?}{=} e(b.c, Y)$. On successful verification, it accepts (outputs 1), else rejects (outputs 0)

- **Revoke** $\mathcal{R}$ is an interactive protocol between the issuer and the user.

  1. The User first sends their non-randomised original credential $\sigma = (a, b, c, d)$ to the issuer.
  2. The Issuer first runs $\mathcal{V}(\sigma, pk)$ to verify whether it is a valid signature, if it fails, the revocation also fails

3. The Issuer then wants to verify if the user knows the secret(α, β) behind their signature, so, the issuer generates $u \xleftarrow{R} \mathbb{Z}_q^*$ and sends $c' = c^u$ and g_1^u to the user.
4. If the user knew the secrets, the user can compute $c'^{\beta^{-1}}$ to get g_1^{xu} and send this back to the issuer, let this be R
5. The issuer simply verifies $e(g_1^u, X = g_2^x) \stackrel{?}{=} e(R = g_1^{xu}, g_2)$ and thus checks if the user has sent the correct value. If not, the revocation fails. (Note: Since this verification part does not require the knowledge of the private key x, revocation can be moved from the issuing authority to a separate revocation authority too)
6. Now, the issuer checks for an entry in T with $A = a$ and $B = b$. If found, that entry is removed and the revocation is done, since that user's credentials will no longer be updated and hence cannot be used from the next epoch.
7. If such an entry was not found in T, it means the user sent a randomised version of their credential and hence the revocation will simply fail.

(Note: that the above modification is for self-revocation only, i.e. the user has to voluntarily participate honestly in the above protocol)

2 Encryption

The Key Generation Mechanism at the RSU and the encryption scheme for further communications by the user in the zone are unchanged from the original scheme.

3 Properties of the Authentication Scheme

- **Correctness of Verification:** Let $sk_{IA} = (x, y), pk_{IA} = (X = g_2^x, Y = g_2^y)$ and $\sigma = (a, b, c = a^x, d = (bc)^y)$.
  - $e(c = a^x, g_2) = e(a, g_2^x) = e(a, X)$
  - $e(d = (bc)^y, g_2) = e((bc), g_2)^y$ And, $e(b.c, Y) = e(bc, g_2^y) = e(bc, g_2)^y$
- **Randomisability and Unlinkability:** The modified scheme retains the randomisability and unlinkability properties of the original scheme.
  If $\sigma = (a, b, c, d)$ is a valid credential in an epoch, and $r \in \mathcal{Z}_q^*$, let $\bar{\sigma'} = (a^r, b^r, c^r, d^r)$. Then,
  - σ' is also a valid credential in that epoch. Since
    - $e(a^r, X) \stackrel{?}{=} e(c^r, g_2)$, $e(a, X)^r \stackrel{?}{=} e(c, g_2)^r$
    - $e(d^r, g_2) \stackrel{?}{=} e(b^r c^r, Y)$, $e(d, g_2)^r \stackrel{?}{=} e(bc, Y)^r$
  - Also, σ' is unlinkable from σ

- **Unforgeability of Credentials:**
  Only the issuing authority can issue a valid credential or update a valid credential of a previous epoch. In the unforgeability experiment, the adversary $\mathcal{A}$ not only has access to an issuing oracle $\mathcal{O}(a, b)$ that can give a valid credential for (a, b), it also has access to the pk_{IA} for the past N epochs and a valid credential for any possible (a, b) in any of those N epochs. So, $\mathcal{A}$ has access to:

  1. $\mathcal{O}_1(a, b)$: returns a valid credential (a, b, c, d) for this epoch
  2. $\mathcal{O}_2(i)$: returns public key X_i of the issuing authority for the epoch i epochs back from the current epoch.
  3. $\mathcal{O}_3(a, b, i)$: returns a credential (a, b, c, d) that was valid at the epoch i epochs back from the current epoch

  Now, the unforgeability experiment is the $\mathcal{A}$ forging a credential on some (a, b) with access to the above three oracles, except that $\mathcal{O}_1$ cannot be queried for the to-be-forged credentials's (a, b) itself.
  Claim 1: $\mathcal{A}$ has negligible chances of succeeding in this unforgeability experiment when N is polynomial in the security parameter.
  Proof: We analyse two cases: when the current epoch's public key X is equal to atleast one of the previous N epochs' public key and the case when it is not equal to any of them

  - **case 1:** $\exists i \in \{1...N\}$ s.t. $X = X_i$
    In this case, $\mathcal{A}$ can simply query $\mathcal{O}_3(a, b, i)$ and the output is a valid forged credential for this epoch.
    $\Rightarrow \mathcal{A}$ succeeds with probability in case 1.
    Probability of case 1 $= Pr[E] \leq \dfrac{N}{\text{Sample space size of } X} = \dfrac{poly(k)}{2^k}$
    (Equality when all the N epochs have distinct X_is)
    Note that $Pr[E].Pr[(\text{success of } \mathcal{A})|E] \leq negl(k)$
  - **case 2:** $\nexists i \in \{1...N\}$ s.t. $X = X_i$
    Probability of case 2 $= Pr[\bar{E}] = 1 - Pr[E]$

    Claim 2: Access to $\mathcal{O}_2$ and $\mathcal{O}_3$ does not provide $\mathcal{A}$ any non-negligible advantage in forging a credential in case 2.
    Proof: $\mathcal{A}$ wants to forge $(a, b, c = a^x, d = (bc)^y)$. Let us focus on forging $c = g_1^{x\beta}$. Adversary knows g_1^{β} and g_2^{x}. Adversary also knows $\{(c_i, d_i)\}$ for $i = 1, .., N$. Since x is not equal to any of x_is, knowing c_is does not help in any way. c does not depend on y. So, If an adversary was capable of forging a correct $c = g_1^{x\beta}$ from g_1^{β} and g_2^{x} and a list of $g_1^{x_i\beta}$ and $g_2^{x_i}$, we can use this adversary to solve a co-CDH problem of finding $g_1^{x\beta}$ from g_1^{β} and g_2^{x} by generating N random x_is and passing these values also to the adversary, which will output a correct $g_1^{x\beta}$, which breaks our hardness assumption. So, no such $\mathcal{A}$ exists.

    (NOTE: THE ABOVE BOX IS THE NON-RIGOROUS OR SLIGHTLY LESS CONFIDENT PART)

Now, with only access to $\mathcal{O}_1$, this experiment reduces to the unforgeability experiment in the original scheme, which we know to have negligible chances of success for the $\mathcal{A}$
$\Rightarrow Pr[\bar{E}].Pr[(\text{success of } \mathcal{A})|\bar{E}] \leq Pr[(\text{success of } \mathcal{A})|\bar{E}] \leq negl(k)$

$Pr[(\text{success of } \mathcal{A})] = Pr[E].Pr[(\text{success of } \mathcal{A})|E] + Pr[\bar{E}].Pr[(\text{success of } \mathcal{A})|\bar{E}]$
$\leq negl(k)$
Thus, no efficient adversary can succeed in forging a credential with non-negiligible chances of success.

- **Only the user can revoke themselves**, i.e. even if an adversary $\mathcal{A}$ knew the credentials of an user, without the knowledge of the secrets of the user, the adversary cannot revoke the user. Since, during revocation, the issuing authority sends $c' = c^u = g_1^{xu\beta}$ and g_1^u to the user and expects back g_1^{xu}. the problem of forging g_1^{xu} from $(U = g_1^u, X = g_2^x)$ can be mapped to the co-CDH instance of computing g_1^{uv} from $(U = g_1^u, V = g_2^v)$.

4 Properties of the Encryption Scheme

Since, the encryption scheme is unchanged, the CPA, CCA security properties are retained in this modified scheme too.