NTNU | Norwegian University of Science and Technology

PROTOCOL APIS

TTM4205 – Lecture 11

Tjerand Silde

12.10.2023

Contents

Contents

Protocol APIs

By this we mean, on a high level, a server that:

Protocol APIs

By this we mean, on a high level, a server that:

- holds secrets where clients can make queries

Protocol APIs

By this we mean, on a high level, a server that:

- holds secrets where clients can make queries
- holds secrets that clients can interact with

Protocol APIs

By this we mean, on a high level, a server that:

- holds secrets where clients can make queries
- holds secrets that clients can interact with

Protocol APIs

By this we mean, on a high level, a server that:

- holds secrets where clients can make queries
- holds secrets that clients can interact with
- combine inputs to verify batches at once

Protocol APIs

We will look at examples where a client can:

- extract secret signing keys
- forge signatures
- trick a verifier

Several of which are similar to the weekly problems.

We will also look at some mitigations to these issues.

Contents

Recap: Schnorr Signatures

Let $\mathbb{G}$ be a group of prime order p and let g be a generator for $\mathbb{G}$. Denote by pp the public parameters $(\mathbb{G}, g, p)$.

Let H be a cryptographic hash function that outputs uniformly random elements in $\mathbb{Z}_p$.

Let the secret key $\mathsf{sk} \leftarrow\$ \mathbb{Z}_p$ be sampled uniformly at random, and let the public key be $\mathsf{pk} = g^{\mathsf{sk}}$, where pk is made public.

Recap: Schnorr Signatures

The Schnorr signature of message m is computed as:

1. Sample random $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and compute $R = g^r$.
2. Compute the output challenge as $c = H(\mathsf{pp}, \mathsf{pk}, m, R)$.
3. Compute the response $z = r - c \cdot \mathsf{sk}$. Output $\sigma = (c, z)$.

To verify the signature, compute $R' = g^z \cdot \mathsf{pk}^c$ and check if $c \stackrel{?}{=} H(\mathsf{pp}, \mathsf{pk}, m, R')$. If correct, accept, and otherwise reject.

Distributed Schnorr Signatures

Assume that two parties P_0 and P_1 wants to compute a joint Schnorr signature. Then P_i does the following:

KGen :

- Sample random $\mathsf{sk}_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and compute $\mathsf{pk}_i = g^{\mathsf{sk}_i}$.
- Send pk_i to party P_{1-i}. Set $\mathsf{pk} = \mathsf{pk}_0 \cdot \mathsf{pk}_1 = g^{\mathsf{sk}_0 + \mathsf{sk}_1}$.

This is called an additive secret sharing of the signing key.

Distributed Schnorr Signatures

Sign:

- Sample random $r_i \leftarrow\$ \mathbb{Z}_p$ and compute $R_i = g^{r_i}$.
- Send R_i to party P_{1-i}. Set $c = H(\mathsf{pp}, \mathsf{pk}, m, R_0 \cdot R_1)$.
- Send the response $z_i = r_i - c \cdot \mathsf{sk}_i$ to party P_{1-i}.

The signature $\sigma = (c, z_0 + z_1)$ can be verified as usual.

Question: How can a malicious client P_0 interacting with an honest (protocol API) P_1 break this signature scheme?

Potential Attacks

- The adversary can control the nonce values
- Repeated nonces for different m's leak sk_1
- (The adversary can bias the secret-public keys)
- (The adversary can abort to deny signatures)
- (All parties need to be online to sign together)

Mitigations in Practice

- Send hashes in an extra round in KGen and Sign
- Send $h_i = H(\mathsf{pk}_i)$ then pk_i and $h'_i = H(R_i)$ then R_i
- (If signatures are deterministic we need other solutions)
- Make it a t-out-of-n signature instead of 2-out-of-2

Proactive Two-Party Signatures for User Authentication

Antonio Nicolosi, Maxwell Krohn, Yevgeniy Dodis, and David Mazières
NYU Department of Computer Science
{nicolosi,max,dodis,dm}@cs.nyu.edu

Figure:
https://www.scs.stanford.edu/~dm/home/papers/nicolosi:2schnorr.pdf

Two-Round Stateless Deterministic Two-Party Schnorr Signatures From Pseudorandom Correlation Functions

Yashvanth Kondi, Claudio Orlandi, and Lawrence Roy

Aarhus University, Aarhus, Denmark
ykondi@cs.au.dk, orlandi@cs.au.dk, ldr709@gmail.com

Figure: https://eprint.iacr.org/2023/216.pdf

Contents

BLS Signatures

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of prime order p with generators g_1, g_2, g_T. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear paring such that $\hat{e}(g_1^a, g_2^b) = g_T^{a \cdot b}$ for all $a, b \in \mathbb{Z}_p$ and H be a cryptographic hash function that outputs uniformly random elements in $\mathbb{G}_2$.

Let the secret key $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$ be sampled uniformly at random, and let the public key be $\mathsf{pk} = g_1^{\mathsf{sk}}$, where pk is made public.

A signature is computed as $\sigma = H(m)^{\mathsf{sk}}$. The verifier checks $\hat{e}(g_1, \sigma) = \hat{e}(\mathsf{pk}, H(m))$. If correct; accept, otherwise reject.

BLS Multisignatures

We can efficiently verify many signatures at once:

- Given many triples $(\mathsf{pk}_i, m_i, \sigma_i)$, compute: $\sigma = \Pi_i \sigma_i$
- Verify all signatures as: $\hat{e}(g_1, \sigma) = \Pi_i \hat{e}(\mathsf{pk}_i, H(m_i))$
- If all messages are identical: $\hat{e}(g_1, \sigma) = \hat{e}(\Pi_i \mathsf{pk}_i, H(m))$
- If the same signers we can aggregate keys: $\mathsf{apk} = \Pi_i \mathsf{pk}_i$

Question: Fix m and pk_0, how can an adversary forge a signature for pk_0 that verifies in the aggregated setting?

Potential Attacks

- Set $\mathsf{pk}_1 = g_1^{\alpha} \cdot (\mathsf{pk}_0)^{-1}$ and signature $\sigma = H(m)^{\alpha}$
- Then $\hat{e}(g_1, \sigma) = \hat{e}(g_1^{\alpha}, H(m)) = \hat{e}(\mathsf{pk}_0 \cdot \mathsf{pk}_1, H(m))$

Mitigations in Practice

- Require a proof for secret key knowledge
- Only aggregate distinct messages each time
- Verify a random linear combination of keys/signatures

Compact Multi-Signatures for Smaller Blockchains

Dan Boneh[1], Manu Drijvers[2,3], and Gregory Neven[2]

[1] Stanford University
dabo@cs.stanford.edu
[2] IBM Research – Zurich
{mdr,nev}@zurich.ibm.com
[3] ETH Zurich

Figure: https://eprint.iacr.org/2018/483.pdf

Contents

DL Parameters

For security of (EC)DH and (EC)DSA, we need to work in prime order (sub-) groups for the discrete logarithm problem to be hard. What happens if this is not the case?

DL Attacks

Recall from earlier that:

- Hardness of DL depends on the divisors p of the order
- We have generic attacks that runs in $\sqrt{p}$ time
- We have sub-exponential attacks for finite field groups

Faulty parameters

What information might leak if:

- The order of the (sub-) group is not prime?
- The element is not in the correct (sub-) group?

Use $g^{\mathsf{sk}} \mod p$ as an example (EC in weekly problems).

Question: How might this happen in practice?

Mitigations in Practice

Always verify:

- given parameters
- input elements
- output elements

Contents

The API must always:

The API must always:

- verify protocol parameters

The API must always:

- verify protocol parameters
- verify API inputs

The API must always:

- verify protocol parameters
- verify API inputs
- check API outputs

The API must always:

- verify protocol parameters
- verify API inputs
- check API outputs
- enforce honest interaction

The API must always:

- verify protocol parameters
- verify API inputs
- check API outputs
- enforce honest interaction
- avoid corner case leakage

The API must always:

- verify protocol parameters
- verify API inputs
- check API outputs
- enforce honest interaction
- avoid corner case leakage
- hinder replay attacks

Questions?