Deploying 2PC ECDSA Signatures in the Wild

RWC 2024
Open Source Cryptography Workshop OSCW 2024

Silence Laboratories

Hey!

- iraklis@silencelaboratories.com
- Done things with Inpher,Parfin,Heliax,ZenGo
- Research with EURECOM, NJIT, UofA, EPFL, Inria
- Head Cryptography/Security Architect @SIL

- Deploying TSS libraries for different stakeholders
- DKLS23+Identifiable abort
- Already behind Metamask as a snap
- Soon as a Google Colab notebook

S:LENCE LABORATORIES

Agenda

- Recap: ECDSA Signatures and MPC 2P ECDSA Sigs
- Sec vs Efficiency
  - FB vuln
  - System solution vs cryptography solution
- Maintaining open source cryptography
  - Challenges
  - Architecture
  - Implementation
- 2PC ECDSA on GCP in one click

Standard Cryptography

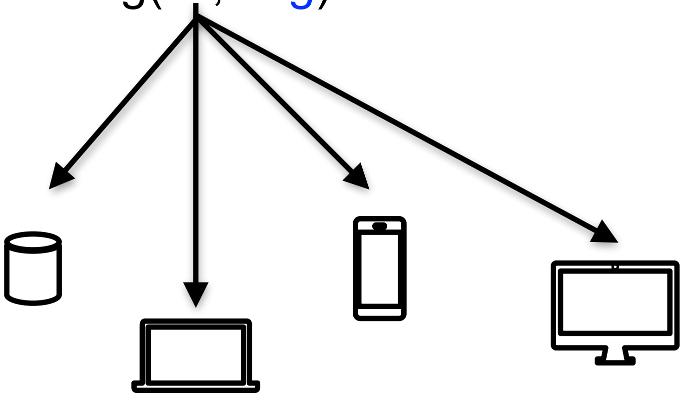
- Encryption: Protect messages end to end
  - PKE: RSA, Paillier, Elgamal: TLS, smart cards
  - Symmetric: AES, ChaCha used daily
- Authentication:
  - Signatures: RSA, ECDSA,Schnorr,EdDSA
  - MAC:HMAC
- KDF:
  - PBKDF: Argon
  - HKDF: HMAC
- Used daily in emails, online banking, smart cards, access control

ECDSA Signature

- Keygen():
  - Choose a secret signing x from an appropriate group
  - Publish your public key pk:= G.x
- Sign(m,x):
  - Choose a random nonce k from an appropriate group
  - Compute R=G.k, take the x coordinate thereof rx
  - Set r = rx
  - Compute s = k.inv (H(m)+x.r)
  - Output r,s
- Verify((r,s),pk,m)
  - Compute a = H(m)/a and b = r/s
  - U = G.a + G.b
  - Let u=(ux.uy)
  - If r==ux accept, otherwise reject

MPC for signatures

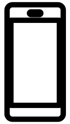
- MPC can compute any function
- Signature computation is a mathematical equation
- Input a secret key and a message
- Sig(sk,msg) = σ



Used to avoid SPOF

2MPC ECDSA - KeyGen

E,G,q



E,G,q

Chose sk1 at random compute $pk1=G^{sk1}$
Chose pk,dk of a PHE

Chose sk2 at random compute $pk2=G^{sk2}$

pk2 ←

$Q=pk2^{sk1}=G^{sk1sk2}$

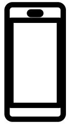
pk1 →

$Q=pk1^{sk}=G^{sk1sk2}$

- Q is the common public ECDSA key
- Q corresponds to sk=sk1*sk2, but nobody knows it in one place
- But still parties can sign under the imaginary sk which verifies to Q

2MPC ECDSA - Sign

 (msg)

 (msg)

Chose k1 at random compute G^{k1}

Chose k2 at random compute G^{k2}

← G^{k2}

$R=G^{k2k1}$

→ G^{k1}

$R=G^{k1k2}$

---

c1 = PHE_pk(H(msg)/k1)

c2 = PHE_pk(rx*sk1/k1)

→ c1,c2

$c3=c1^{-k2}$=PHE_pk(H(msg)/k1$)^{-k2}$=PHE_pk(H(msg)/k)

$c4=c2^{sk2/k2}$=PHE_pk(rx*sk1)/k1$)^{sk2/k2}$= PHE_pk(rx*sk)/k)

c5=c3*c4=PHE_pk(H(msg)/k)*PHE_pk(rx*sk)/k =

PHE_pk(H(msg)+rx*sk)/k)

← c5

PHD_dk(c5)=PHD_dk(PHE_pk(H(msg)+rx*sk)/k)=H(msg)+rx*sk)/k

Outputs sig=(rx=x coordinate of R,s)=H(msg)+rx*sk)/k

Sign

- c1,c2,c3 = do_some_cryptography()

/ecdsa/sign/first - id,c1

- s1,s2 = do_some_cryptography()
- Store in t1 id:c1
- Store in t2 id:s2

s1

- c4 = do_some_cryptography(m,c2,c3,s1)

/ecdsa/sign/second - id,m,c4

- Get from t2 id:s2
- Get from t1 id:c1
- sig = do_some_cryptography(m,c4,s2,c1)
- If ECDSA.verify(sig) == correct then send sig
- Else fail

Sig

Practical Key-Extraction Attacks in Leading MPC Wallets

Nikolaos Makriyannis* Oren Yomtov* Arik Galansky*

January 29, 2024

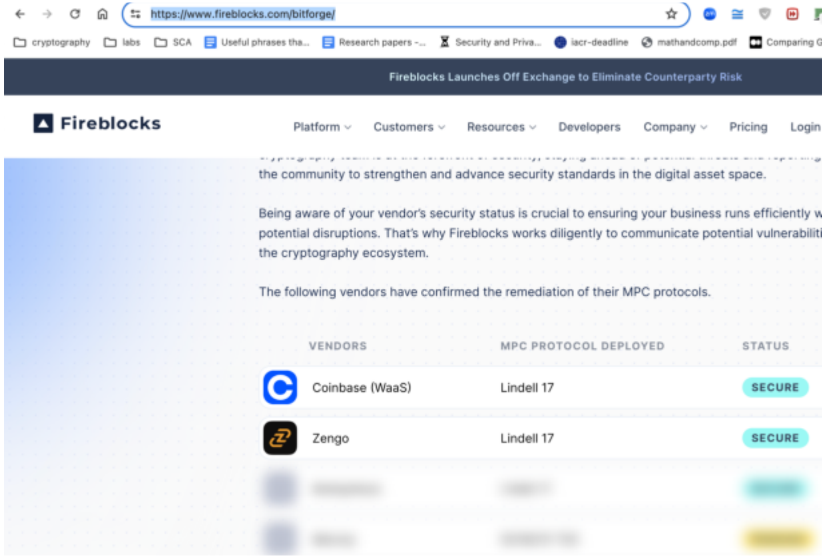
Fb attack

- Exploit the last step of sign/second.
- Sending client side multiple malformed c4’s and from the binary result :success/fail sig, adv could extract one bit at a time and finally recover only server secret share x1 entirely.
- In practise 256 signing rounds where fail happens at each 0 bit
- The problem is that at the last step the server should abort execution per paper but code wasn’t aborting.
- Failing signatures in theory cannot occur from non-malicious clients

- https://www.fireblocks.com/blog/lindell17-abort-vulnerability-technical-report

Attack Takeover

- https://github.com/coinbase/waas-sdk-react-native – prior to version 1.0.0
- https://github.com/ZenGo-X/gotham-city / https://github.com/ZenGo-X/multi-party-ecdsa – prior to tag v1.0.0 (https://github.com/ZenGo-X/gotham-city/releases/tag/v1.0.0)



Abort - System Mitigation

- c1,c2,c3 = do_some_cryptography()

/ecdsa/sign/first - id,c1

If id in
Abort db block
sign

- s1,s2 = d… graphy()
- Store in t1 id:c1
- Store in t2 id:s2

s1

- c4 = do_some_cryptography(m,c2,c3,s1)

/ecdsa/sign/second - id,m,c4

- Get from t2 id:s2
- Get from t1 id:c1
- sig = do_some_cryptography(m,c4,s2,c1)
- If ECDSA.verify(sig) == correct then send sig
- Else fail

If fail mark
that id in the db

Sig

Cryptography mitigation

- P2 still learns the bits up until the first fail
- Is there a better solution?
- ZKP for the correct structure of round 1 msg from P2
- That will put extra 100-150ms

3 Mitigation

Recall that $\mathcal{P}_2$ calculates $c_3 = (1 + u_1 \cdot N) \cdot c_{key}^{u_2} \cdot v^N \mod N^2$, where $u_1 = [k_2^{-1}\mathcal{H}(m) \mod q] + \rho q$ and $u_2 = x_2 \cdot r \cdot k_2^{-1} \mod q$ and $v \leftarrow [N]$.

3.1 ZK Proof

Consider the relation $\mathcal{R}$ that consists of tuples $(C, c_{key}, N; u_1, u_2, v)$ such that

$$C = (1 + u_1 \cdot N) \cdot c_{key}^{u_2} \cdot v^N \mod N^2$$

and (u_1, u_2) are small (say smaller than 2^{700}). The standard sigma protocol for the relation $\mathcal{R}$ goes as follows:

1. Prover sends $D = (1 + \alpha_1 \cdot N) \cdot c_{key}^{\alpha_2} \cdot \beta^N \mod N^2$ for $\alpha_1, \alpha_2 \leftarrow [2^{800}]$ and $\beta \leftarrow [N]$
2. Verifier replies with $e \leftarrow \{0, 1\}$
3. Prover returns (z_1, z_2, w) such that

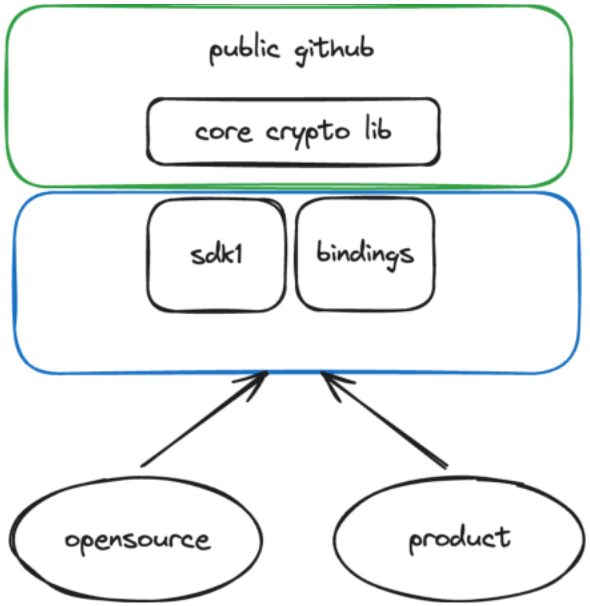
$$\begin{cases} z_1 = \alpha_1 + eu_1 \\ z_2 = \alpha_2 + eu_2 \\ w = \beta \cdot v^N \mod N \end{cases}$$

Verification:

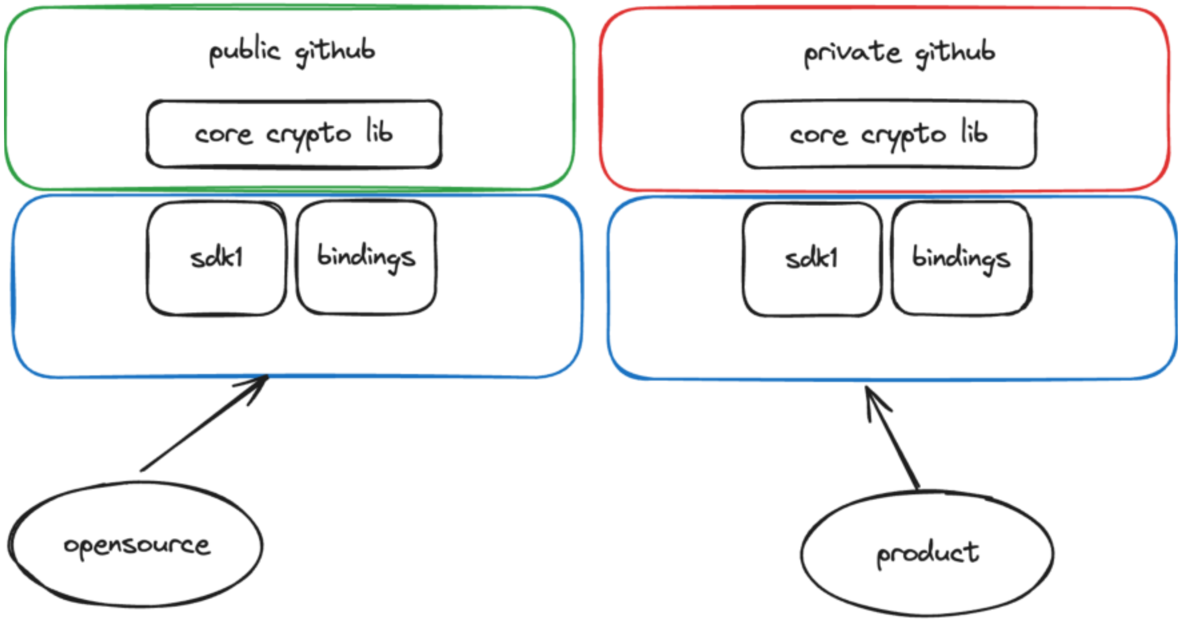
- Check that $u_1, u_2 \in \pm 2^{800}$
- Check that $(1 + z_1 \cdot N) \cdot c_{key}^{z_2} \cdot w^N = D \cdot C^e \mod N^2$

https://github.com/silence-laboratories/rust-2-party-ecdsa

Ideal Cryptography Stack Exposure



Reality

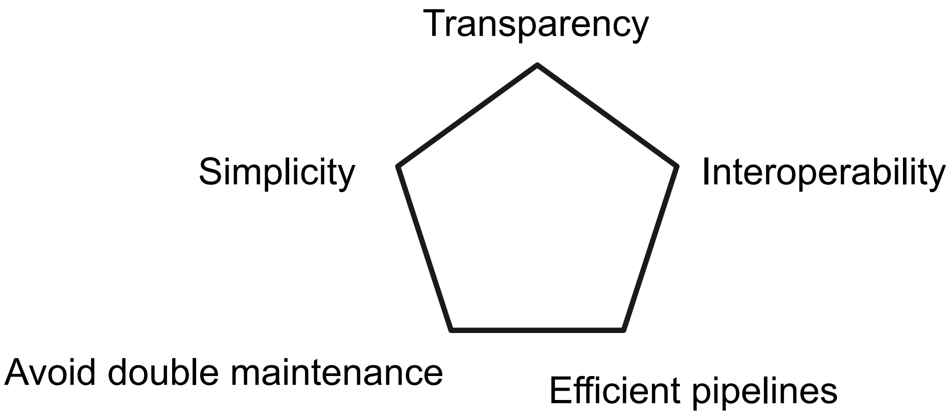


public github
core crypto lib
sdk1
bindings
opensource
private github
core crypto lib
sdk1
bindings
product

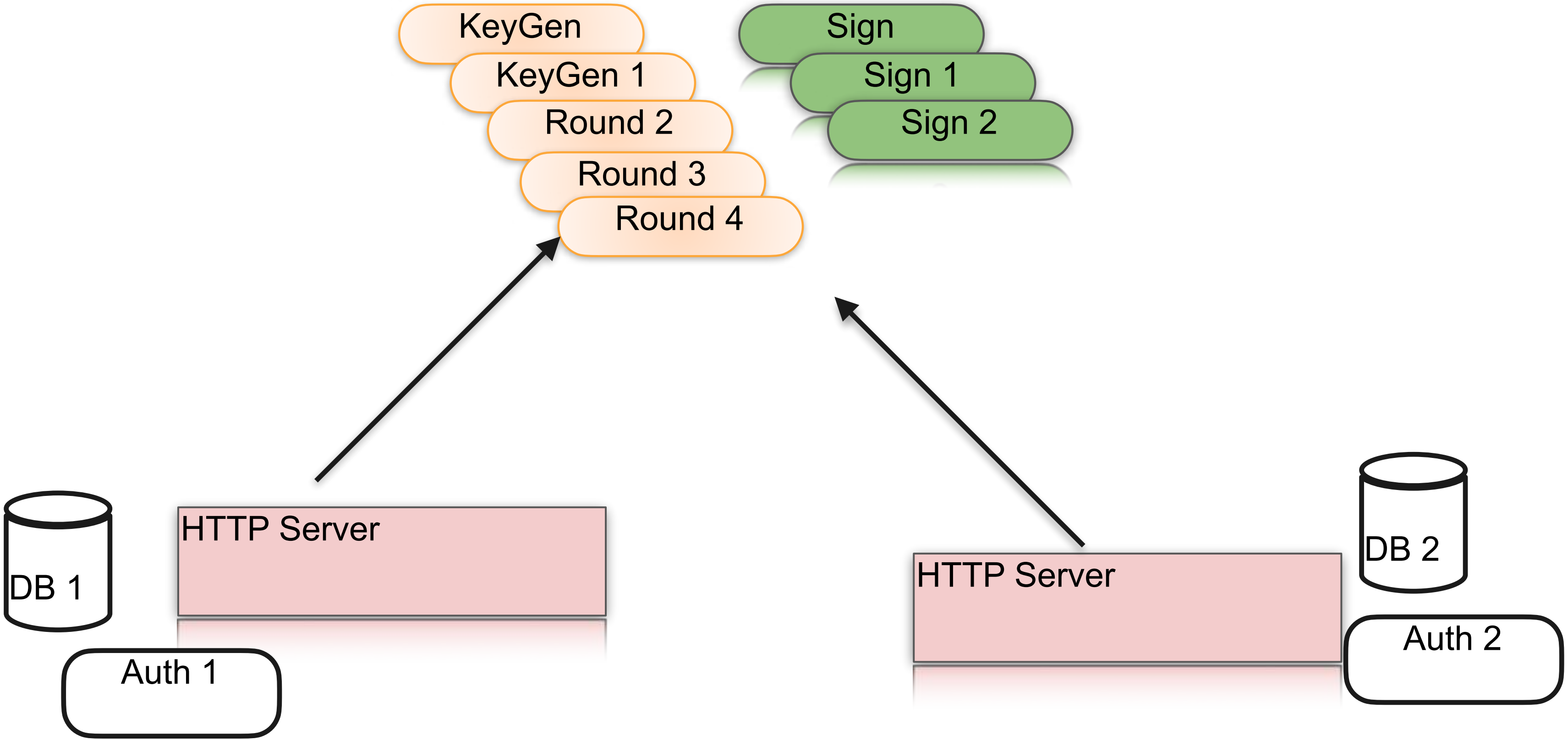
Issues

- How to maintain open source repos being used in production?
- Clients want everything open sourced
- Stakeholders do not want to open source everything or not all the parts
- Maintaining private and public repos becomes challenging:
  - Slow pipelines
  - Duplicate code
  - Not transparent

Open source cryptography stack goals



Abstract Architecture



Abstracting

```
/// The Db trait allows different DB's to implement a
common API for insert and get
#[async_trait]
pub trait Db: Send + Sync {
   async fn insert(
        &self,
        key: &DbIndex,
        table_name: &dyn MPCStruct,
        value: &dyn Value,
    ) -> Result<(), DatabaseError>;
async fn get(
        &self,
        key: &DbIndex,
        table_name: &dyn MPCStruct,
    ) -> Result<Option<Box<dyn Value>>, DatabaseError>;
    async fn has_active_share(&self, customerId: &str)
-> Result<bool, String>;

    /// the granted function implements the logic of tx
authorization. If no tx authorization is needed the
function returns always true
    fn granted(&self, message: &str, customer_id: &str)
-> Result<bool, DatabaseError>;
```

Defaulting cryptographic endpoints

```
#[async_trait]
pub trait KeyGen {
    ///first round of Keygen
 async fn first(
        state: &State<Mutex<Box<dyn Db>>>,
        claim: Claims,
    ) -> Result<Json<(String, KeyGenFirstMsg)>, String> {…code…}
 async fn second(
        state: &State<Mutex<Box<dyn Db>>>,
        claim: Claims,
        id: String,
        dlog_proof: Json<DLogProof>,
    ) -> Result<Json<party1::KeyGenParty1Message2>, String> {…code
async fn third(
        state: &State<Mutex<Box<dyn Db>>>,
        claim: Claims,
        id: String,
        party_2_pdl_first_message: Json<party_two::PDLFirstMessage>,
    ) -> Result<Json<party_one::PDLFirstMessage>, String> {…code…}
 async fn fourth(
        state: &State<Mutex<Box<dyn Db>>>,
        claim: Claims,
        id: String,
        party_two_pdl_second_message: Json<party_two::PDLSecondMessage>,
    ) -> Result<Json<party_one::PDLSecondMessage>, String> {…code…}
```

```
#[async_trait]
pub trait Sign {
    async fn sign_first(
        state: &State<Mutex<Box<dyn Db>>>,
        claim: Claims,
        id: String,
        eph_key_gen_first_message_party_two: Json<party_two::
    ) -> Result<Json<party_one::EphKeyGenFirstMsg>, String> {
  async fn sign_second(
        state: &State<Mutex<Box<dyn Db>>>,
        claim: Claims,
        id: String,
        request: Json<SignSecondMsgRequest>,
    ) -> Result<Json<party_one::SignatureRecid>, String> {…co
```

Wrap Default Impl

- Most http servers in rust ecosystem do not allow mount directly default imps
- Another layer of abstraction is needed

```
#[post("/ecdsa/keygen/first", format = "json")]
pub async fn wrap_keygen_first(
    state: &State<Mutex<Box<dyn Db>>>,
    claim: Claims,
) -> Result<Json<(String, KeyGenFirstMsg)>, String> {
    struct Gotham {}
    impl KeyGen for Gotham {}
    Gotham::first(state, claim).await
}

#[post("/ecdsa/keygen/<id>/second", format = "json", data = "<dlog_proof>")]
pub async fn wrap_keygen_second(
    state: &State<Mutex<Box<dyn Db>>>,
    claim: Claims,
    id: String,
    dlog_proof: Json<DLogProof>,
) -> Result<Json<party1::KeyGenParty1Message2>, String> {
    struct Gotham {}
    impl KeyGen for Gotham {}
    Gotham::second(state, claim, id, dlog_proof).await
}
```

Mounting 2MPC ECDSA Server

```
pub struct PublicGotham {
    rocksdb_client: rocksdb::DB,
}
impl KeyGen for PublicGotham {}

impl Sign for PublicGotham {}
```

HTTP Server

```
pub fn get_server() -> Rocket<Build> {
    let x = PublicGotham::new();
    rocket::Rocket::build()
        .register("/", catchers![internal_error, not_found, bad_request])
        .mount(
            "/",
            routes![
                gotham_engine::routes::wrap_keygen_first,
                gotham_engine::routes::wrap_keygen_second,
                gotham_engine::routes::wrap_keygen_third,
                gotham_engine::routes::wrap_keygen_fourth,
                gotham_engine::routes::wrap_chain_code_first_message,
                gotham_engine::routes::wrap_chain_code_second_message,
                gotham_engine::routes::wrap_sign_first,
                gotham_engine::routes::wrap_sign_second,
            ],
        )
        .manage(Mutex::new(Box::new(x) as Box<dyn gotham_engine::traits::Db>))
}
```

https://github.com/ZenGo-X/gotham-engine

Takeover

- TSS protocols are running in real world
- Transparency
- Abstracting through traits, dyn trait objects
- MPC is not a panacea
- It brings complexity - we can improve

Google Colab Notebook 2MPC ECDSA

- White label a 2 party ECDSA wallet between GCP server and your phone
  - Download on your android: https://drive.google.com/file/d/1jT6NIQBqMO_qB1EwH5UN9PRm99a7yO0D/view?usp=drive_link
  - https://gcsdemo.silencelaboratories.com

- 
SilentShard
Demo

iraklis@silencelaboratories.com

https://github.com/silence-laboratories/dkls23-rs
https://snaps.metamask.io/snap/npm/silencelaboratories/silent-shard-snap/