CryptoExperts
WE INNOVATE TO SECURE YOUR BUSINESS

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

CRY.ME : un challenge de cryptographie sur une messagerie sécurisée

Sonia Belaïd[2], Ryad Benadjila[1,2], **Thibauld Feneuil**[2], Jérémy Jean[1], Louiza Khati[1], Ange Martinelli[1], Chrysanthi Mavromati[1], **Abdul Rahman Taleb**[2] and Matthieu Rivain[2]

[1] ANSSI, [2] CryptoExperts

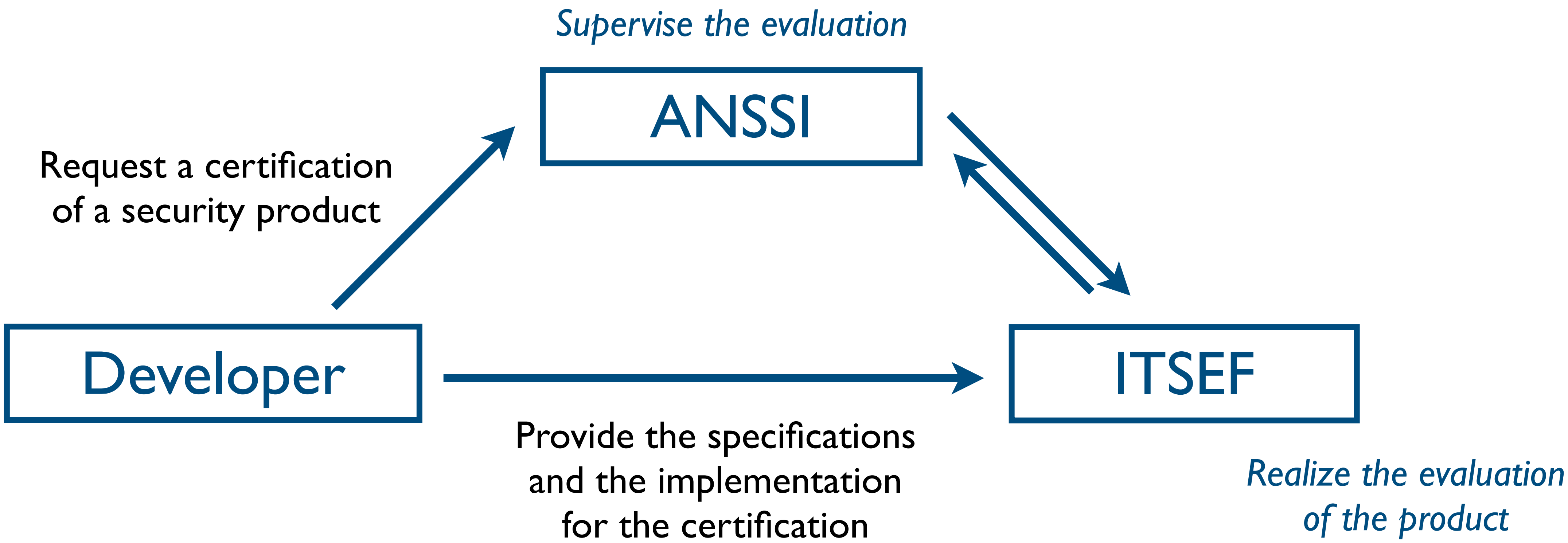
GDR Sécurité Informatique - 28 juin 2023

Designing CRY.ME (CRYptographic MEssaging application)

Why CRY.ME?

ANSSI - French certification center

- Implements the certification scheme
- Licenses the ITSEFs (Information Technology Security Evaluation Facilities) which conduct the evaluations
- Supervises the evaluation projects



Why CRY.ME?

2022: CRY.ME challenge for all ITSEFs and only for cryptographic analysis

- Cryptographic analysis compliant to ANSSI's procedures[1, 2]
  - *Theoretical analysis*: review of the specifications
    - Choice of the primitives, design of the protocols, key management
  - *Validation of the implementation*:
    - Conformity of the implementation
    - Advanced analysis of the code

[1] ANSSI-CC-CRY-P-01 : https://www.ssi.gouv.fr/uploads/2014/11/anssi-cc-cry-p-01-modalites-pour-la-realisation-des-analyses-cryptographiques_v4.1.pdf

[2] ANSSI-PG-083 : https://www.ssi.gouv.fr/uploads/2021/03/anssi-guide-mecanismes_crypto-2.04.pdf

Challenge Design

- Cryptographic analysis compliant to ANSSI's procedures[1, 2]
- Test vehicle common to all ITSEFs (software and hardware) but not only

[1] ANSSI-CC-CRY-P-01 : https://www.ssi.gouv.fr/uploads/2014/11/anssi-cc-cry-p-01-modalites-pour-la-realisation-des-analyses-cryptographiques_v4.1.pdf

[2] ANSSI-PG-083 : https://www.ssi.gouv.fr/uploads/2021/03/anssi-guide-mecanismes_crypto-2.04.pdf

CRYPTOEXPERTS RÉPUBLIQUE FRANÇAISE Liberté Égalité Fraternité

Challenge Design

- Cryptographic analysis compliant to ANSSI's procedures[1, 2]
- Test vehicle common to all ITSEFs (software and hardware) but not only
    - **Open-source software product**

[1] ANSSI-CC-CRY-P-01 : https://www.ssi.gouv.fr/uploads/2014/11/anssi-cc-cry-p-01-modalites-pour-la-realisation-des-analyses-cryptographiques_v4.1.pdf

[2] ANSSI-PG-083 : https://www.ssi.gouv.fr/uploads/2021/03/anssi-guide-mecanismes_crypto-2.04.pdf

CryptoExperts RÉPUBLIQUE FRANÇAISE Liberté Égalité Fraternité

Challenge Design

- Cryptographic analysis compliant to ANSSI's procedures[1, 2]
- Test vehicle common to all ITSEFs (software and hardware) but not only
  - **Open-source software product**
- Cover vulnerabilities of different types and levels of difficulty
  - **Several types of vulnerabilities:** conformance, symmetric crypto, asymmetric crypto, random number generation, protocols, implementation
  - **Three levels of difficulty:** not necessarily the same difficulty for identification and exploitation
  - **Various sources of errors:** weak choice of algorithms, design and/or implementation errors, differences between specifications and implementation, etc.

[1] ANSSI-CC-CRY-P-01 : https://www.ssi.gouv.fr/uploads/2014/11/anssi-cc-cry-p-01-modalites-pour-la-realisation-des-analyses-cryptographiques_v4.1.pdf

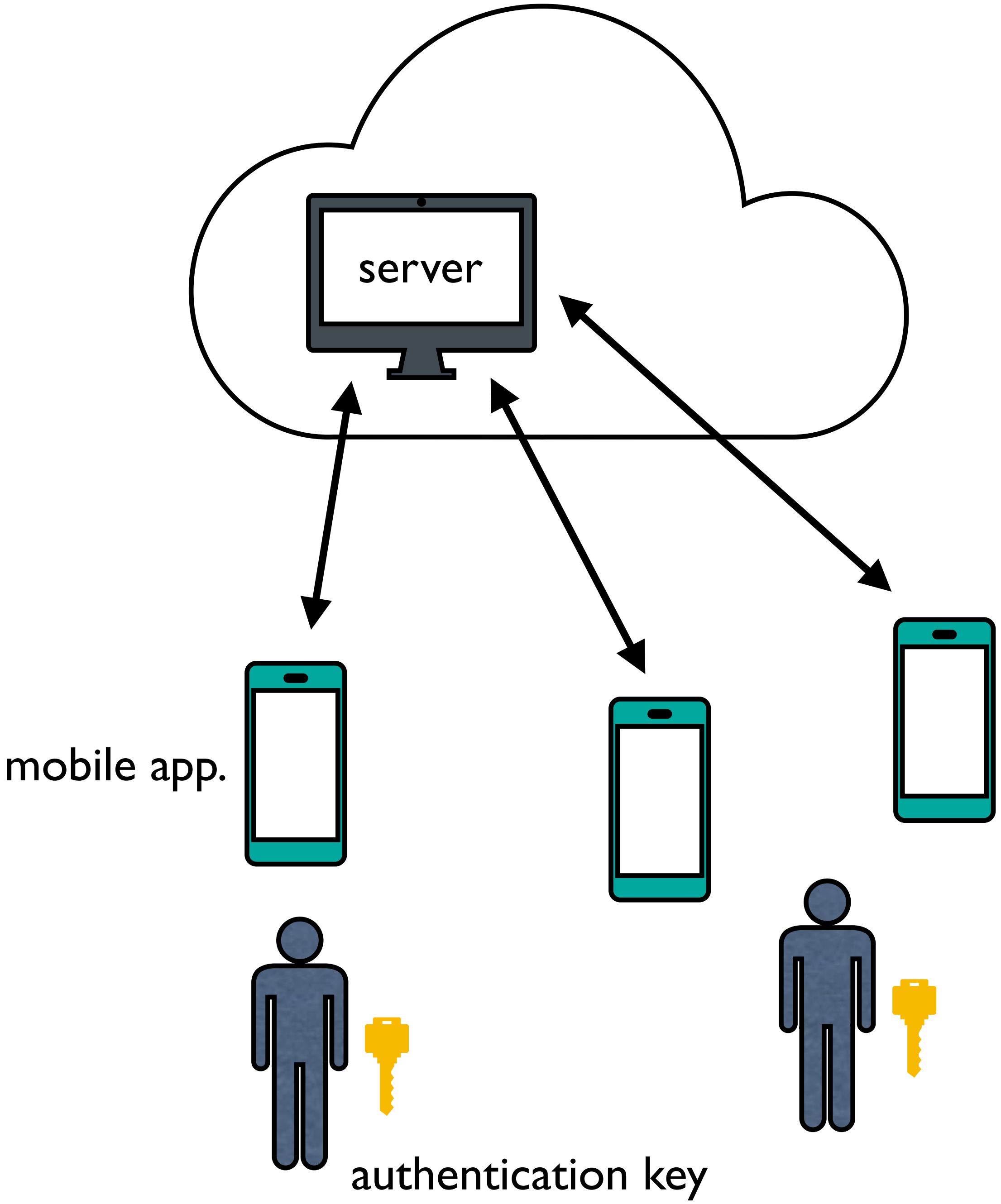
[2] ANSSI-PG-083 : https://www.ssi.gouv.fr/uploads/2021/03/anssi-guide-mecanismes_crypto-2.04.pdf

CRY.ME « Element » Application

Why Element?

- Open-source Android application
- Integrates Matrix end-to-end encryption
- Integrates secure data backup
- Supports group conversations
- Convenient code modification and navigation (Yubikey integration, …)
- Easy-to-use Interface
- No external dependencies besides matrix android SDK

Architecture



Functionalities

- Create account
- Login / Logout
- Send / Receive messages (1-to-1 or group)
- Send / Receive attachments
- Secure backup storage on the server
- Out-of-band verification between users

Github Repo

- Open-source code: https://github.com/ANSSI-FR/cry-me
- **Modified** Element code with/without vulnerability comments
- Server code running in a Docker container
- Emulator code running on a desktop and handling authentication keys and server connections
- Documents (in French) describing security specifications and targets

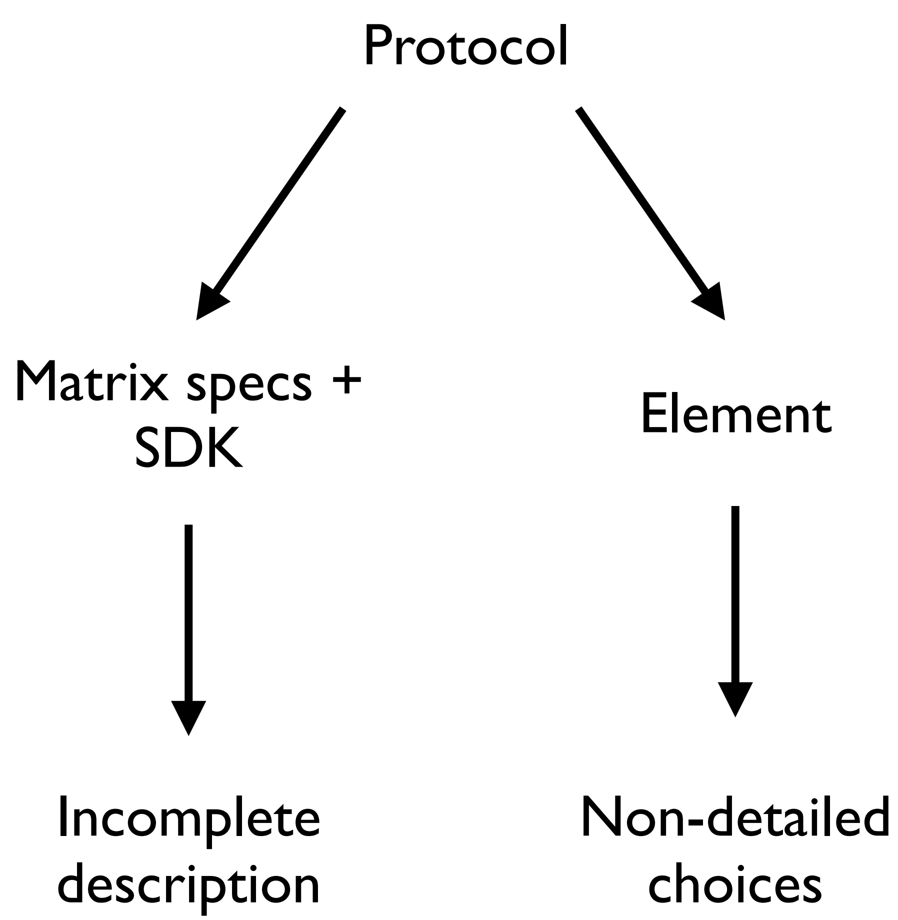
Challenge Setup

Setup of Different Layers

Front-end ↓ *Back-end*

Layer	Language	Action
Application (UI)	Android Kotlin	Integrate Yubikey Adapt Add vulnerabilities
Matrix SDK (protocols) / Yubikit SDK	Kotlin Java	Adapt Add vulnerabilities
Olm SDK (crypto. functions)	Java	Adapt
Crypto. API	C++	Adapt
Crypto. library	C	Re-implement Add vulnerabilities

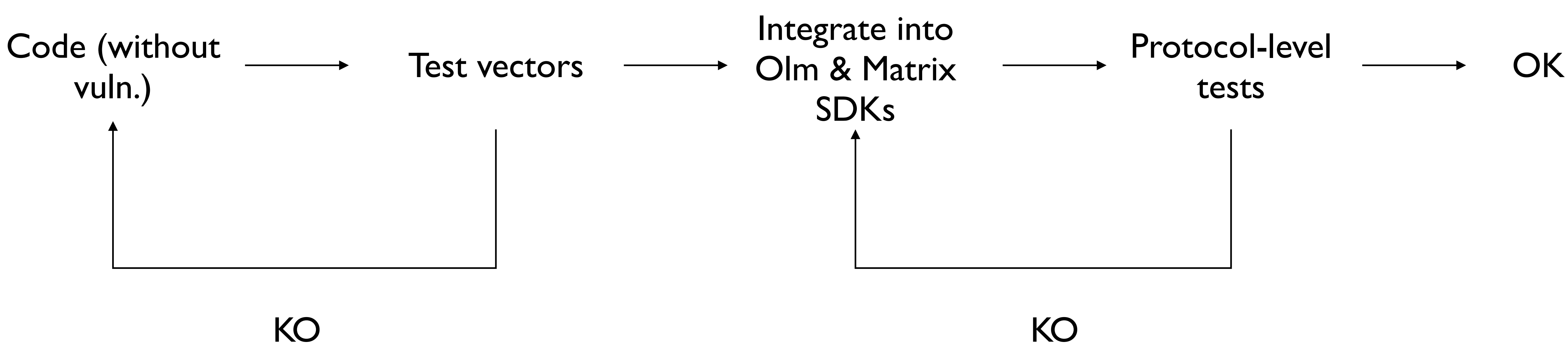
Cryptographic Protocols



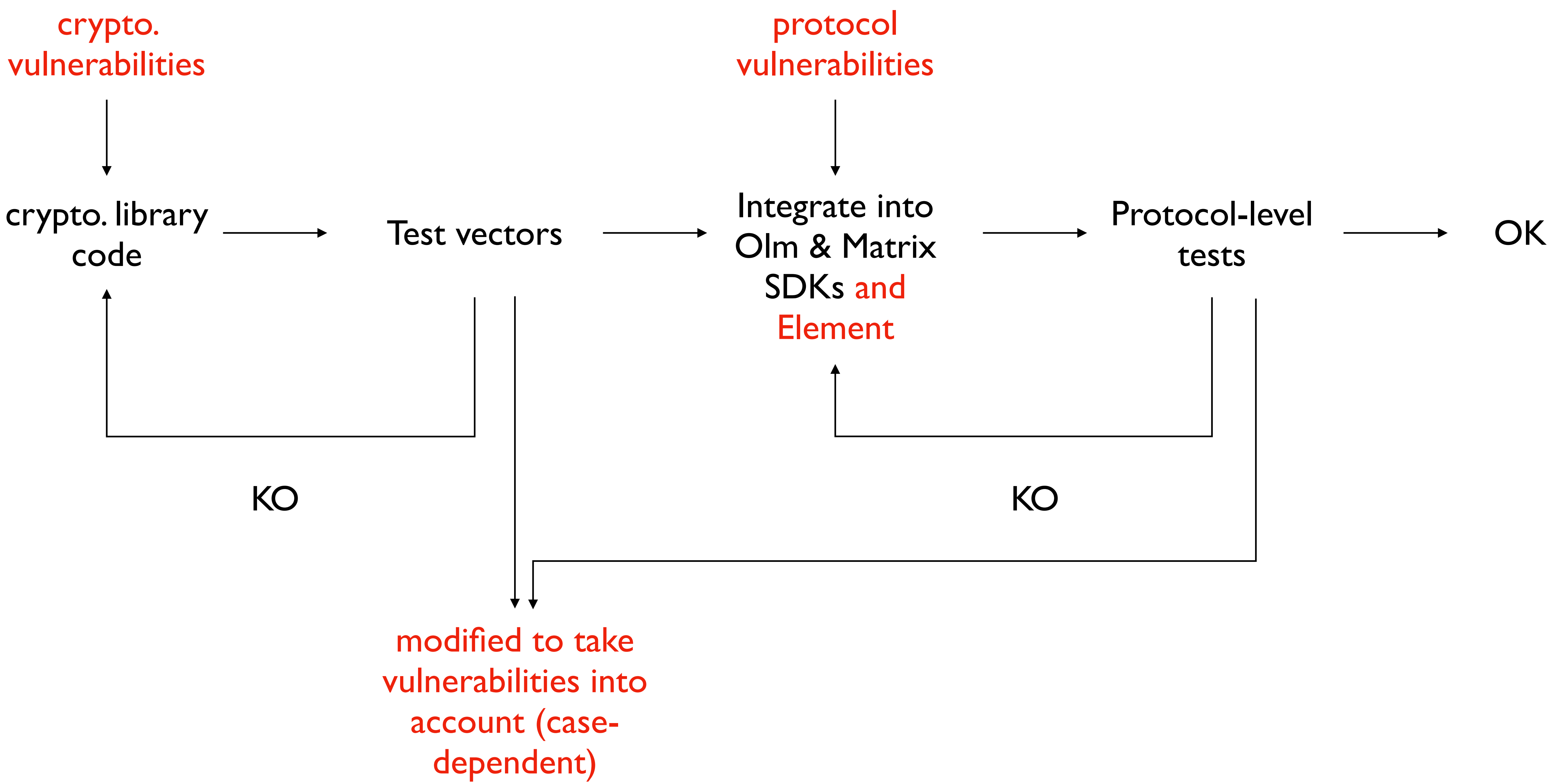
- Goal: provide a complete comprehensive description from both sources for each protocol

Implementation of Cryptographic Primitives

- Matrix uses Olm SDK, implementing all crypto. algorithms
- CRY.ME: re-implementing all crypto. algorithms from scratch
- Process must ensure application’s functionality



Integration of Vulnerabilities



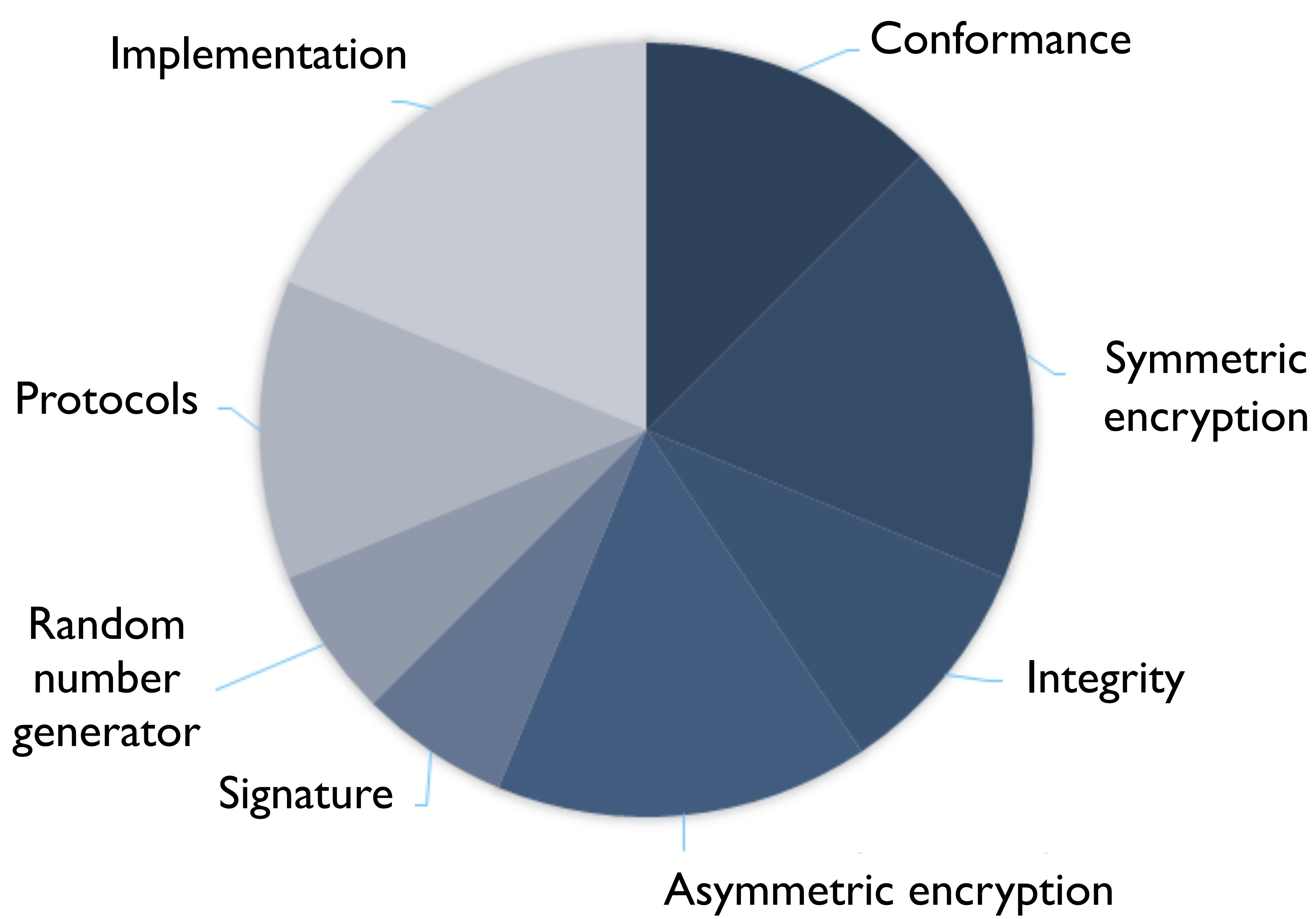
Server Setup

- 3-layer architecture
- Nginx
  - handles TLS
- Synapse
  - handles app. functionalities
- PostgreSQL
  - handles database

A Docker container with the server setup is available in the git repository.

Vulnerabilities

Vulnerabilities



Choice of Cryptographic Primitives

	Original library	New library
Symmetric encryption	AES, RC4, Blowfish, ROT-13, DES	AES
Hash	SHA256, MD2, MD5, SHA1	SHA1, SHA3
Mac	HMAC	HMAC
Key derivation	HKDF	PBKDF2, HKDF
Key exchange	ECDH on Curve25519	ECDH on Wei25519
Signature	Ed25519	RSA, Wei25519
Pseudo-random generator	Java built-in secure PRG	ECC PRG

Example 1: AES-256

- Pseudocode from FIPS 197 standard

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])              // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                          // See Sec. 5.1.1
      ShiftRows(state)                         // See Sec. 5.1.2
      MixColumns(state)                        // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Nb = 4, Nr = 14

Example 1: AES-256

- CRY.ME implementation

```
// --- AES BLOCK FUNCTIONS
// ------------------------------------------------------------------------------

void aes256_encrypt(const BYTE in[], BYTE out[], const WORD exp_key[])
{
    BYTE state[AES_BLOCK_SIZE];
    int r;

    // init state
    memcpy(state, in, AES_BLOCK_SIZE);

    add_round_key(state, exp_key);

    for(r=0; r<NB_ROUNDS; r++)
    {
        sub_bytes(state);
        shift_rows(state);
        mix_columns(state);
        add_round_key(state, exp_key+r*4);
    }

    sub_bytes(state);
    shift_rows(state);
    add_round_key(state, exp_key+NB_ROUNDS*4);

    // output final state
    memcpy(out, state, AES_BLOCK_SIZE);
}
```

Example 1: AES-256

- CRY.ME implementation

```c
// --- AES BLOCK FUNCTIONS
// ------------------------------------------------------------------------------

void aes256_encrypt(const BYTE in[], BYTE out[], const WORD exp_key[])
{
    BYTE state[AES_BLOCK_SIZE];
    int r;

    // init state
    memcpy(state, in, AES_BLOCK_SIZE);

    add_round_key(state, exp_key);

    for(r=0; r<NB_ROUNDS; r++)
    {
        sub_bytes(state);
        shift_rows(state);
        mix_columns(state);
        add_round_key(state, exp_key+r*4);
    }

    sub_bytes(state);
    shift_rows(state);
    add_round_key(state, exp_key+NB_ROUNDS*4);

    // output final state
    memcpy(out, state, AES_BLOCK_SIZE);
}
```

Incorrect output on test vectors!

Example 1: AES-256

- CRY.ME implementation

```c
// --- AES BLOCK FUNCTIONS
// -----------------------------------------------------------------------------

void aes256_encrypt(const BYTE in[], BYTE out[], const WORD exp_key[])
{
    BYTE state[AES_BLOCK_SIZE];
    int r;

    // init state
    memcpy(state, in, AES_BLOCK_SIZE);

    add_round_key(state, exp_key);

    for(r=0; r<NB_ROUNDS; r++)
    {
        sub_bytes(state);
        shift_rows(state);
        mix_columns(state);
        add_round_key(state, exp_key+r*4);
    }

    sub_bytes(state);
    shift_rows(state);
    add_round_key(state, exp_key+NB_ROUNDS*4);

    // output final state
    memcpy(out, state, AES_BLOCK_SIZE);
}
```

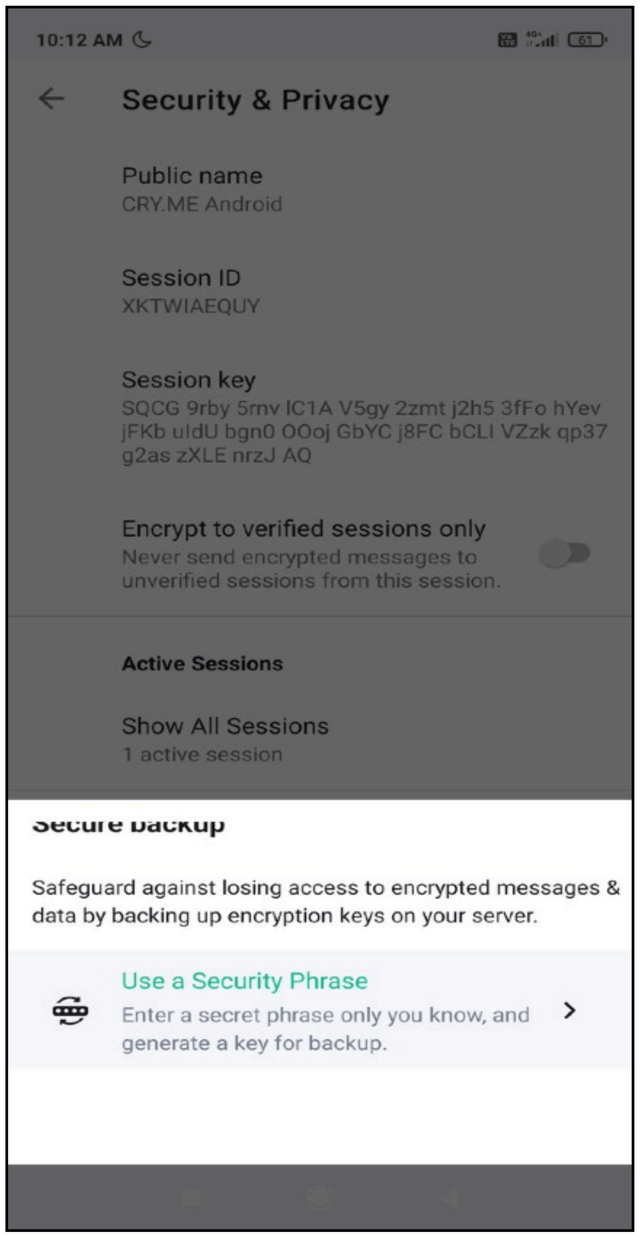
Incorrect output on test vectors!

Example 1: AES-256

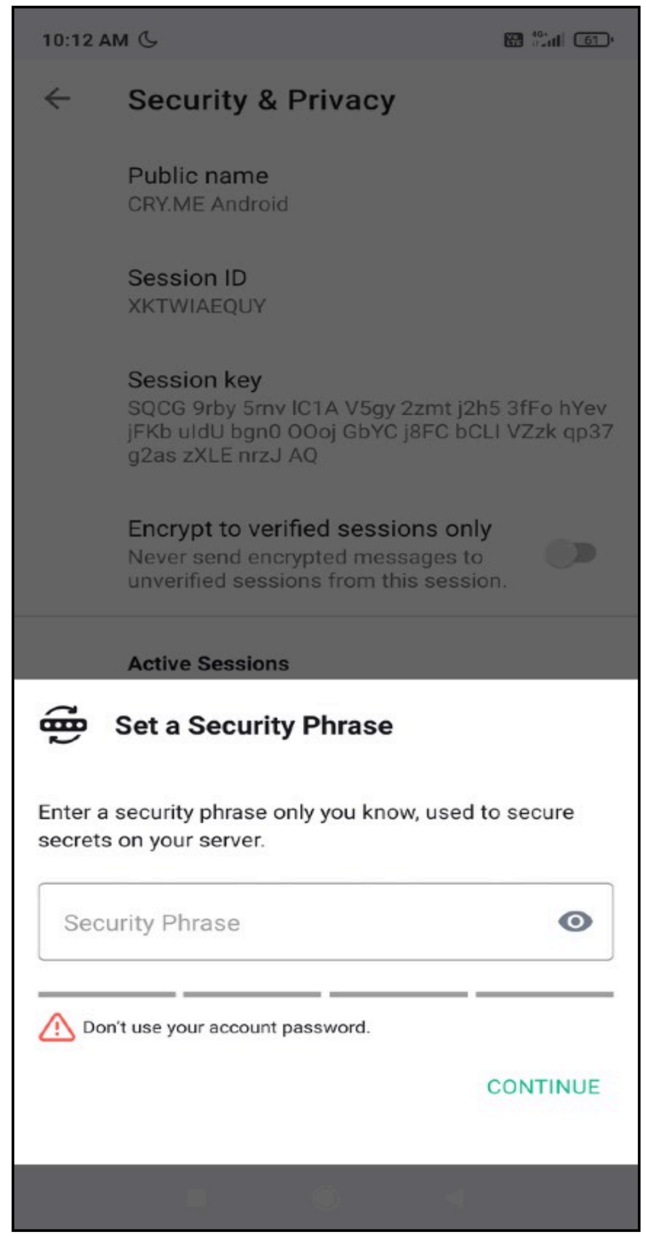
- AES implementation that does not respect the standard
  - Type: conformity issue
  - Identification: easy
  - Exploitation: impossible

Example II: PBKDF2

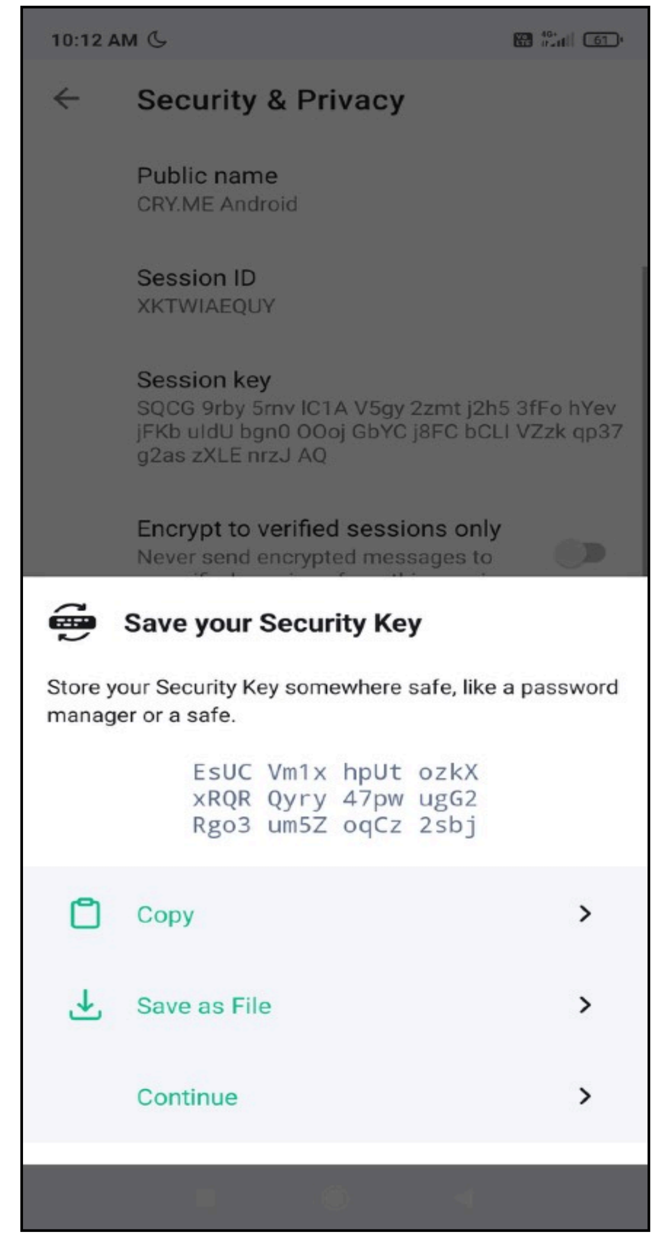
- PBKDF2 generates key for secret storage, using a passphrase



I- select option for passphrase

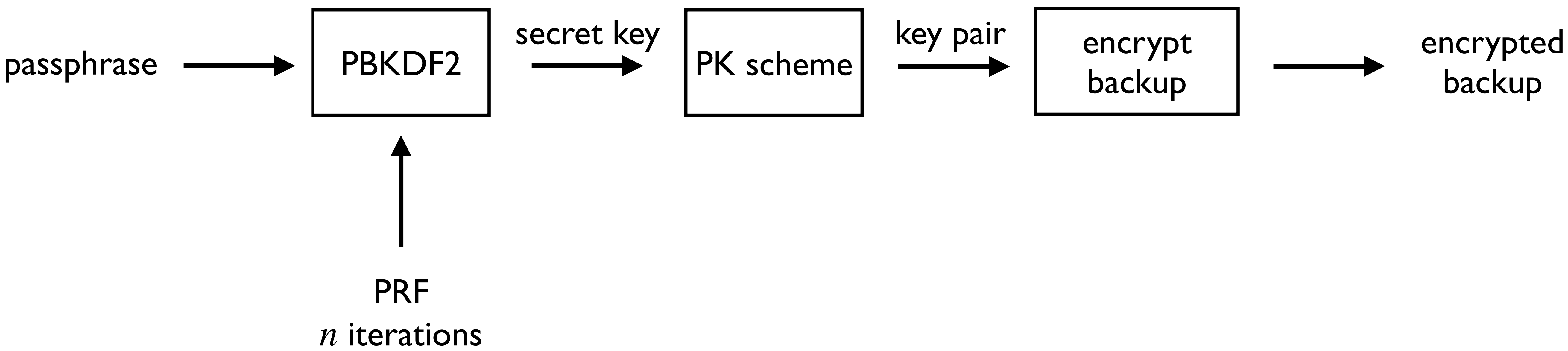


II - choose passphrase



III - backup successfully generated

Example II: PBKDF2



Typically $n = 2^{20}$

Example II: PBKDF2

- PBKDF2 implementation in C
- 2^{20} iterations

```c
/** default number of iterations for pbkdf2 algorithm */
#define DEFAULT_PBKDF2_ITERATIONS (1 < 20)
```

Example II: PBKDF2

- PBKDF2 implementation in C
- 2^{20} iterations

```c
/** default number of iterations for pbkdf2 algorithm */
#define DEFAULT_PBKDF2_ITERATIONS (1 < 20)
```

$= 1$

should have been $(1 << 20)$

Developer error!

Example II: PBKDF2

- Somewhere in PBKDF2 implementation …

```c
for(uint32_t i = 1; i<(l+1); i++){
    get_four_bytes(i, S_concat+salt_length);

    compute_hmac(sha_type, password, 4, S_concat, salt_length+4, T);
    memcpy(U, T, hlen*sizeof(uint8_t));

    for(uint32_t k=1; k<c; k++){
        compute_hmac(sha_type, password, 4, U, hlen, U);

        for(uint32_t j = 0; j< hlen; j++){
            T[j] ^= U[j];
        }
    }

    if(i == l){
        memcpy(DK + ((i-1)*hlen), T, r*sizeof(uint8_t));
    }else{
        memcpy(DK + ((i-1)*hlen), T, hlen*sizeof(uint8_t));
    }
}
```

Example II: PBKDF2

- Somewhere in PBKDF2 implementation …

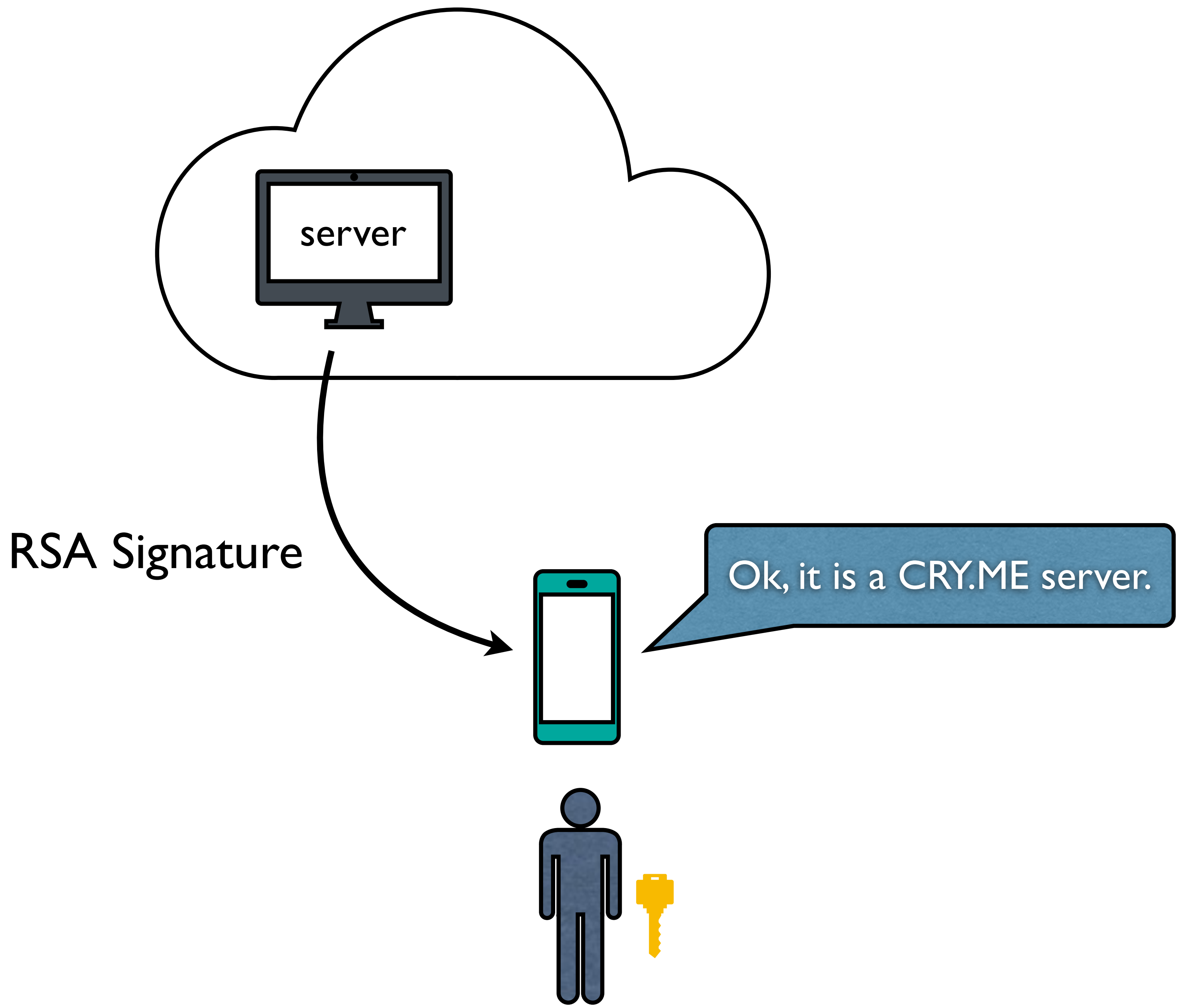
```
    for(uint32_t i = 1; i<(l+1); i++){
        get_four_bytes(i, S_concat+salt_length);

        compute_hmac(sha_type, password, 4, S_concat, salt_length+4, T);
        memcpy(U, T, hlen*sizeof(uint8_t));

        for(uint32_t k=1; k<c; k++){
            compute_hmac(sha_type, password, 4, U, hlen, U);

            for(uint32_t j = 0; j< hlen; j++){
                T[j] ^= U[j];
            }
        }

        if(i == l){
            memcpy(DK + ((i-1)*hlen), T, r*sizeof(uint8_t));
        }else{
            memcpy(DK + ((i-1)*hlen), T, hlen*sizeof(uint8_t));
        }
    }
```

Uses only 4 bytes of the password!

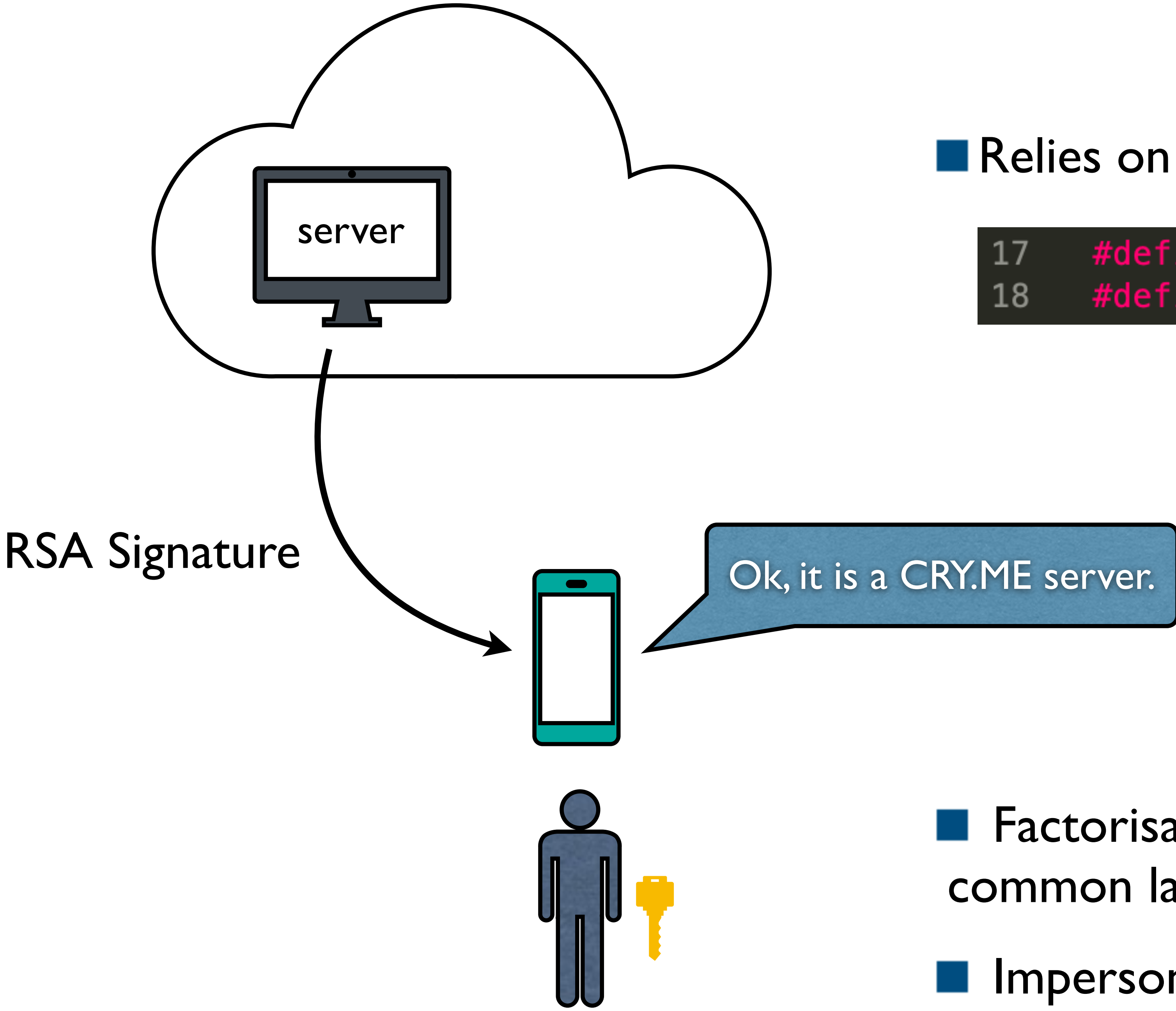
Example II: PBKDF2

- 2 vulnerabilities leading to an exploitable attack on secure backup storage
- Brute force search for the secret key with 2^{32} possible passwords
  - Type: implementation bug
  - Identification: medium
  - Exploitation: easy

Example III: RSA Signature



Example III: RSA Signature

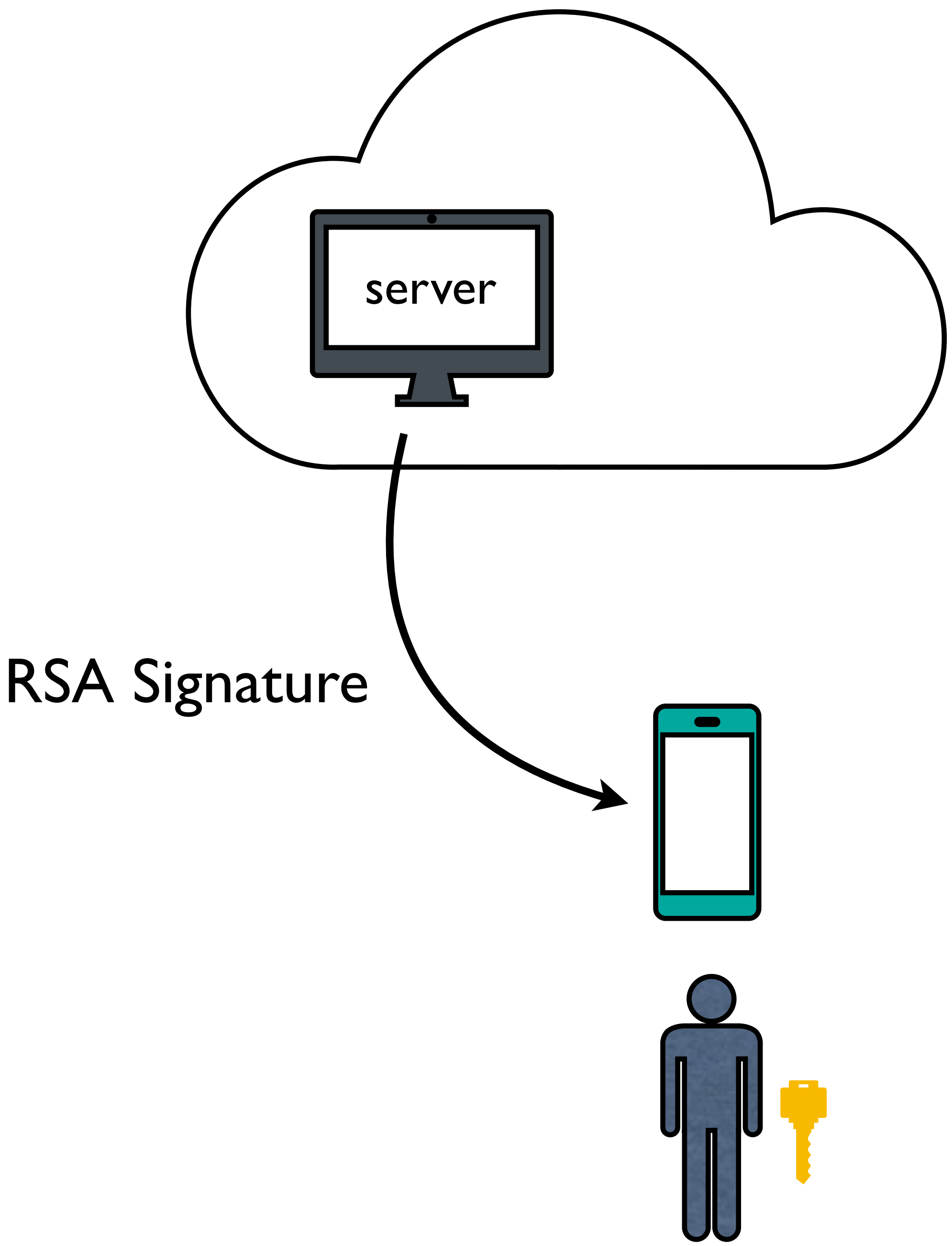


- Relies on RSA-512

```
#define RSA_ACCREDITATION_MOD_LENGTH 64
#define RSA_ACCREDITATION_HASH_LENGTH 20
```

- Factorisation takes ~10 days using a common laptop
- Impersonates server's identity

Example III: RSA Signature



- Rely on RSA-512

```
#define RSA_ACCREDITATION_MOD_LENGTH 64
#define RSA_ACCREDITATION_HASH_LENGTH 20
```

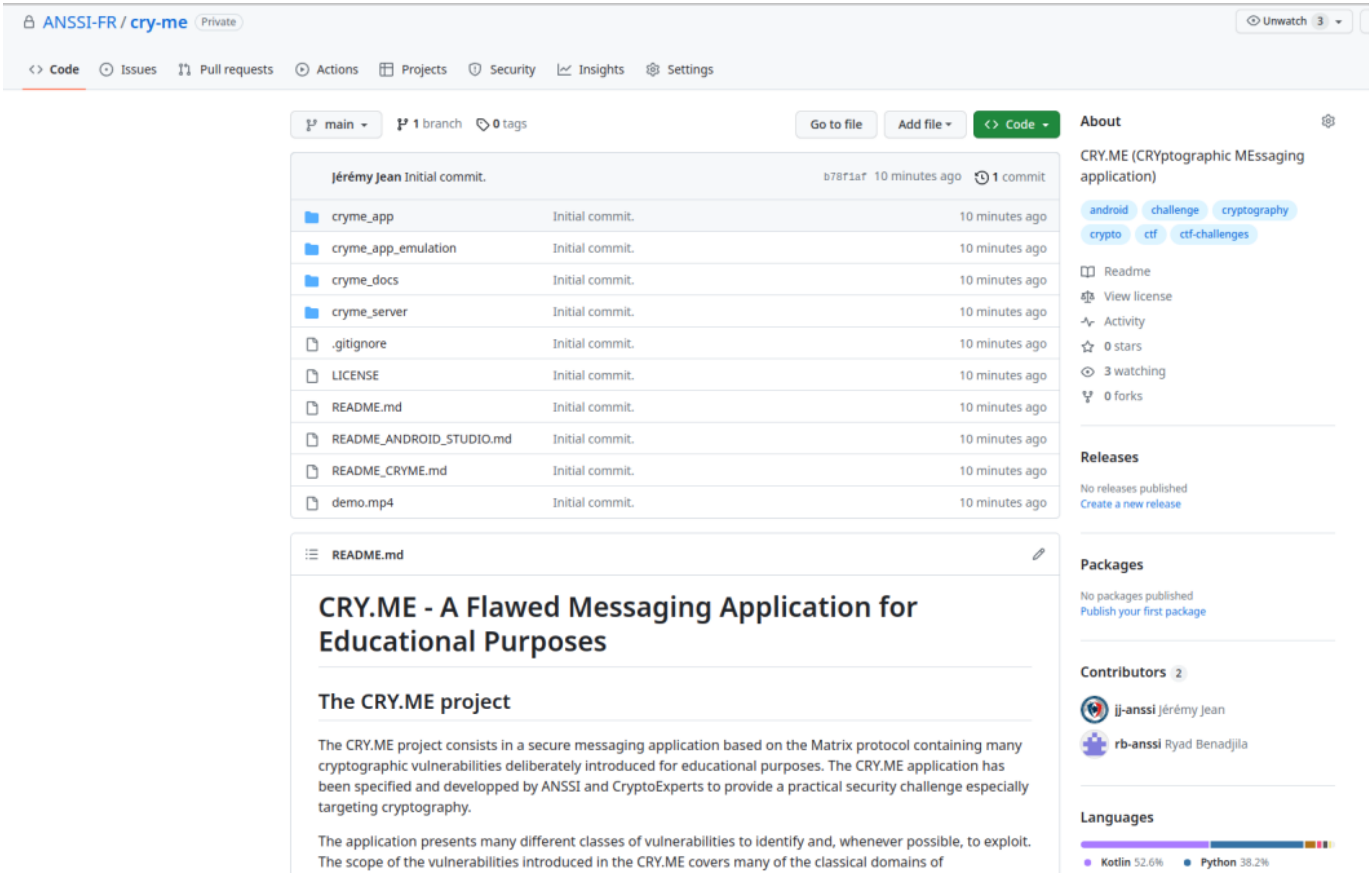
  - Type: asymmetric encryption
  - Identification: medium
  - Exploitation: medium

- Factorisation takes ~10 days using a common laptop
- Impersonates server's identity

Conclusion

- Many **cryptographic vulnerabilities** to find
- **Open-source** material with a guided tutorial
  - https://github.com/ANSSI-FR/cry-me
- The challenge serves as a tool for **learning purposes** on
  - the impact of cryptographic vulnerabilities on a known messaging application
  - how to detect them
  - if and how to exploit them

Questions ?



CRYPTOEXPERTS

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité