McEliece in practice

NATO Workshop on Secure Implementation of PQC

Secure implementation of post-quantum cryptography
SPS Project Number: 984520

O. Grošek, V. Hromada, **Pavol Zajac**

Institute of Computer Science and Mathematics
Slovak University of Technology

September 26, 2016

Outline

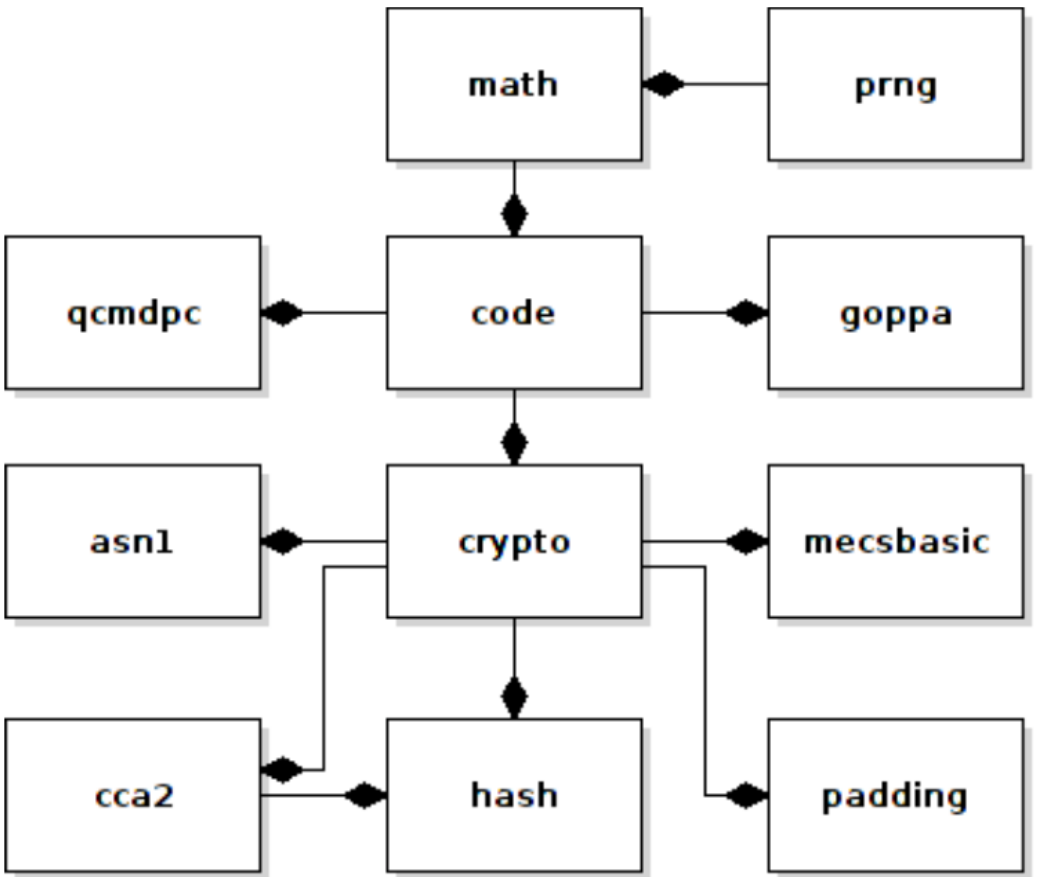
Software implementations

Various student projects:

- Main version of BitPunch:
  `https://github.com/FrUh/BitPunch`
- Crypto-box extension:
  `https://github.com/n0whereman/cryptobox`
- LDGM signatures based on BitPunch:
  `https://github.com/schwarzwald/dp2015`
- Other implementations:
  - Android McEliece messenger based on BouncyCastle
  - Standalone AVR QC-MDPC implementation

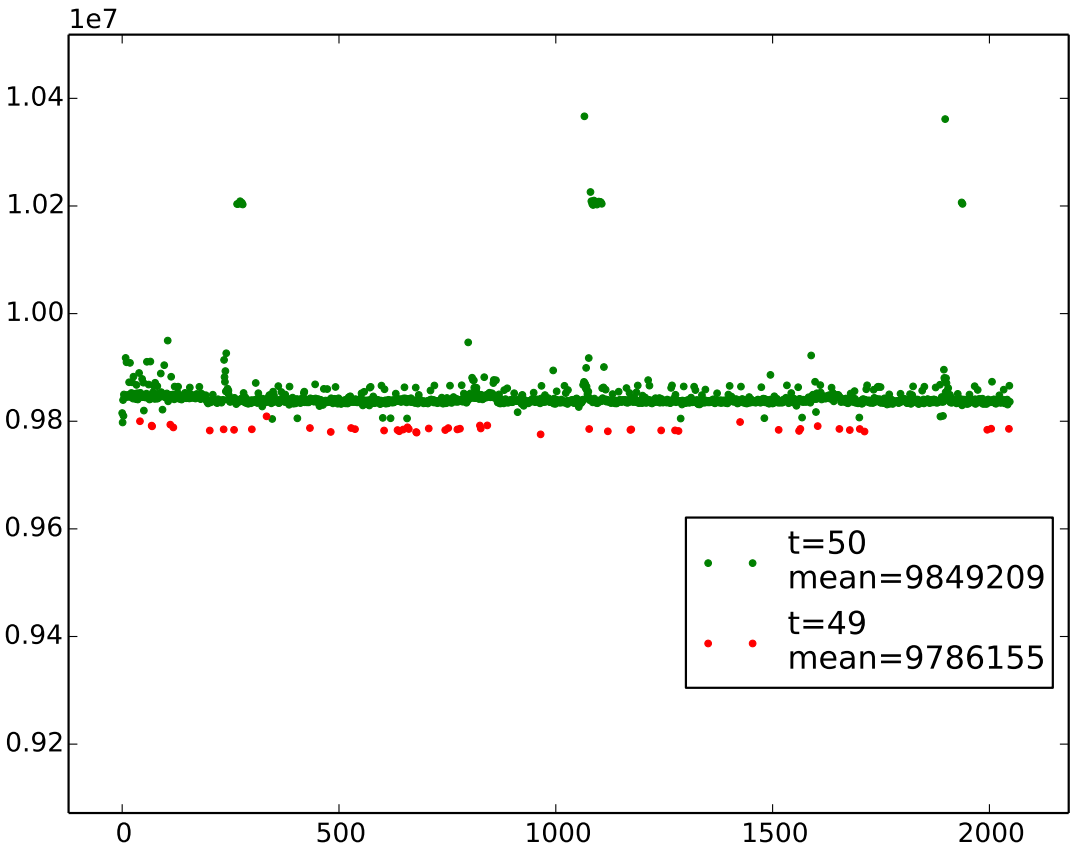
BitPunch

Standalone, modular C implementation of McEliece cryptosystem, developed at UIM FEI STU



Side Channel Attacks resistance

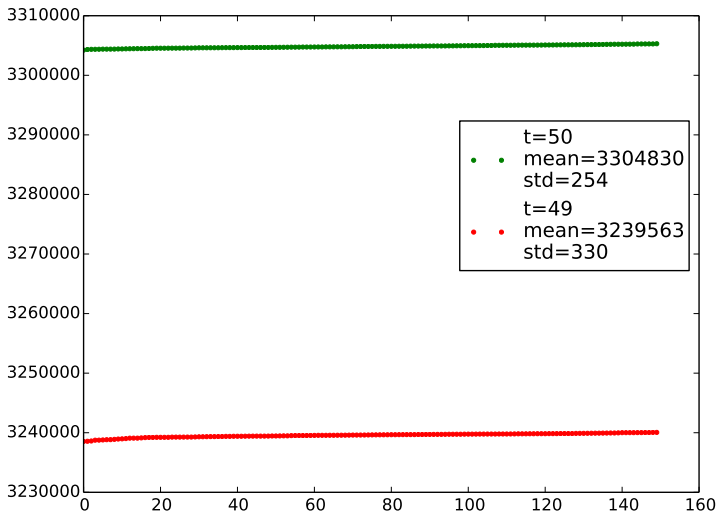
- Original BitPunch: vulnerable to timing based reaction attack



SCA countermeasures

- Resistance to Error Locator Polynomial attack:
  - Constant-time multiplication in finite field
  - Constant-time evaluation of polynomials
  - Evaluation of ELP $\approx$ 2.5 times slower
- Open problem: efficient constant time XGCD
- Open problem: cache timing effects

SCA resistance



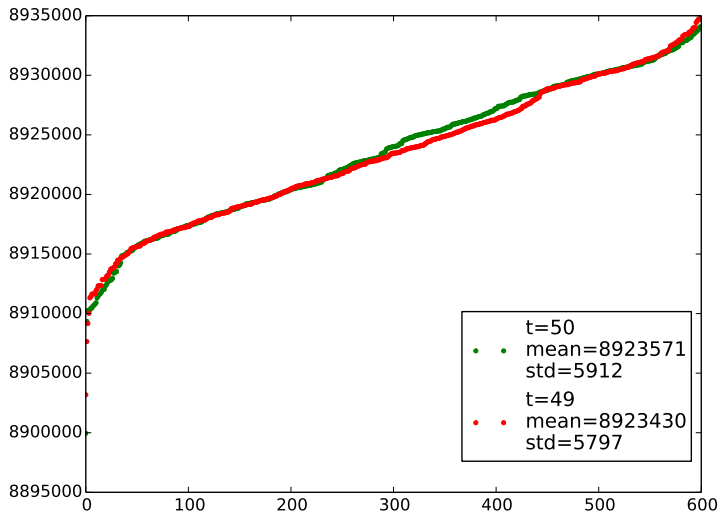


Figure: Evaluation of polynomials without (left) and with (right) countermeasures.

BitPunch extensions and modules

- Internal code choice: Goppa or QC-MDPC
- Experimental signatures (standalone): LDGM
- Cryptobox (standalone)

CCA2 security

Original McEliece is not secure:

- Known partial-plaintext reduces computational costs
- Related-message / message-resend attack
- Reaction attack
- Message malleability

Solution: CCA2-secure conversion (padding)

Incorrect version of Pointcheval's conversion

- Potential problem with systematic public key $G = (I|R)$
- Original CCA2 conversion in BitPunch (from Shoufan et.al. 2010):

$$\begin{array}{ccccc} y_1|| & y_2|| & y_3|| & y_4|| & y_5 \\ k_e \oplus e_1|| & hash(m||k_i) \oplus e_2|| & \tilde{m} \cdot R \oplus e_3|| & m \oplus hash(k_e)|| & k_i \oplus hash(e) \end{array}$$

- y_1, y_4 — "symmetric" encryption
- y_2, y_5 — integrity of ciphertext
- y_3 — McEliece encryption overhead

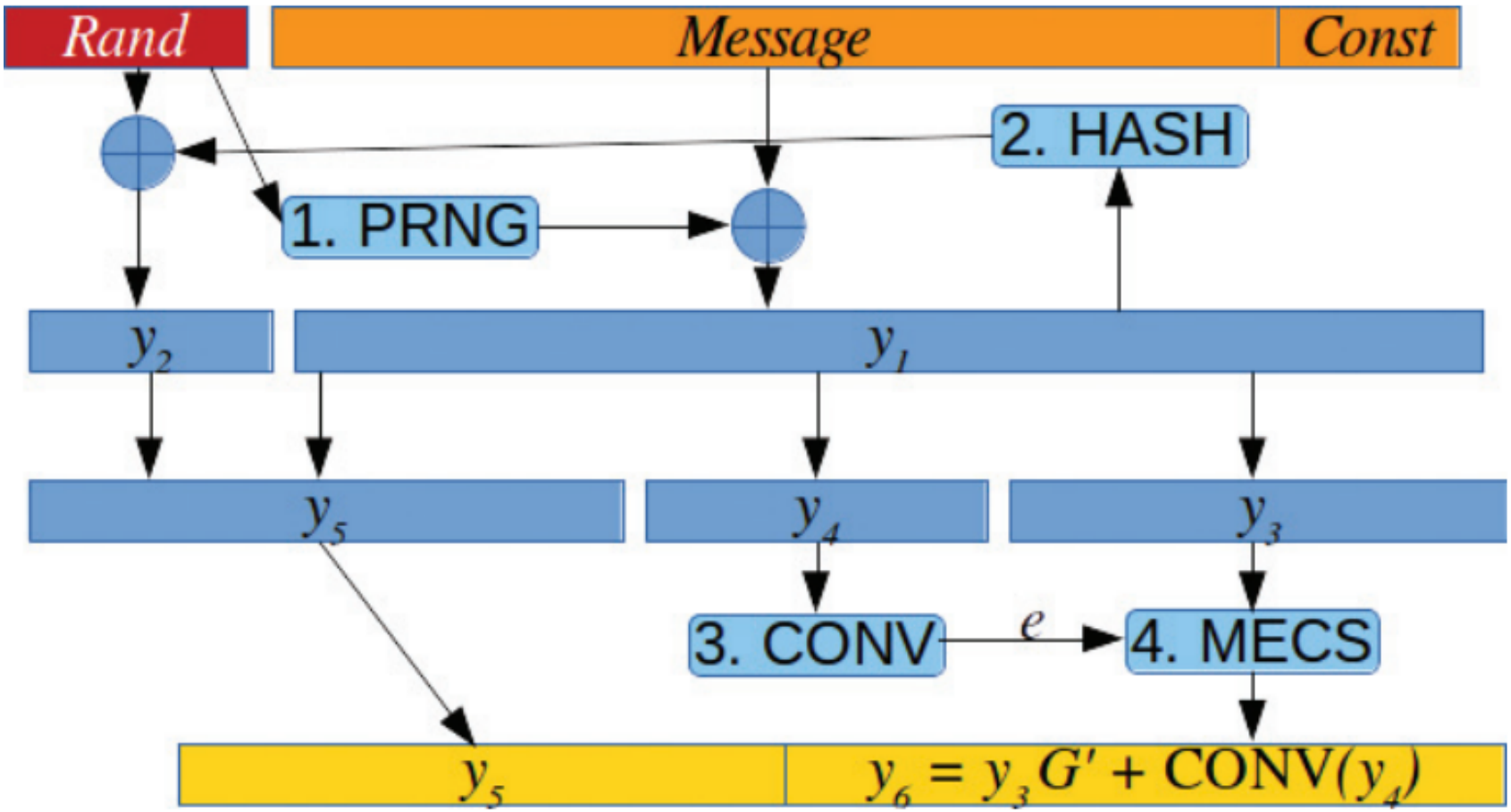
Incorrect version of Pointcheval's conversion

- Potential problem with systematic public key $G = (I|R)$
- Original CCA2 conversion in BitPunch (from Shoufan et.al. 2010):

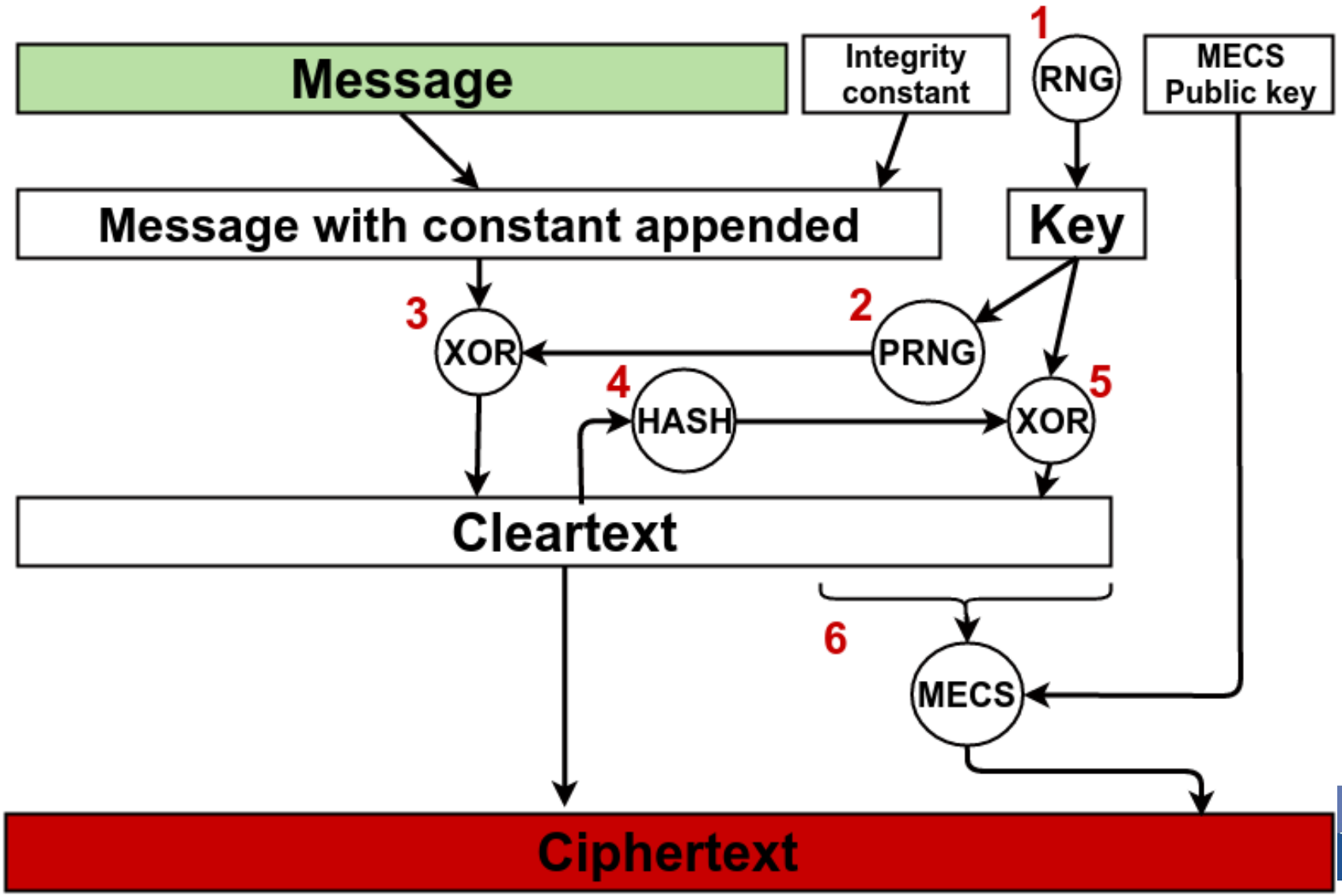
$$\begin{array}{ccccc} y_1 || & y_2 || & y_3 || & y_4 || & y_5 \\ k_e \oplus e_1 || & hash(m||k_i) \oplus e_2 || & \tilde{m} \cdot R \oplus e_3 || & m \oplus hash(k_e) || & k_i \oplus hash(e) \end{array}$$

- ATTACK: only find errors in y_1 part, instead of $y_1||y_2||y_3$.
- Security paradox: Longer hashes lead to a weaker system.

Kobara-Imai conversion

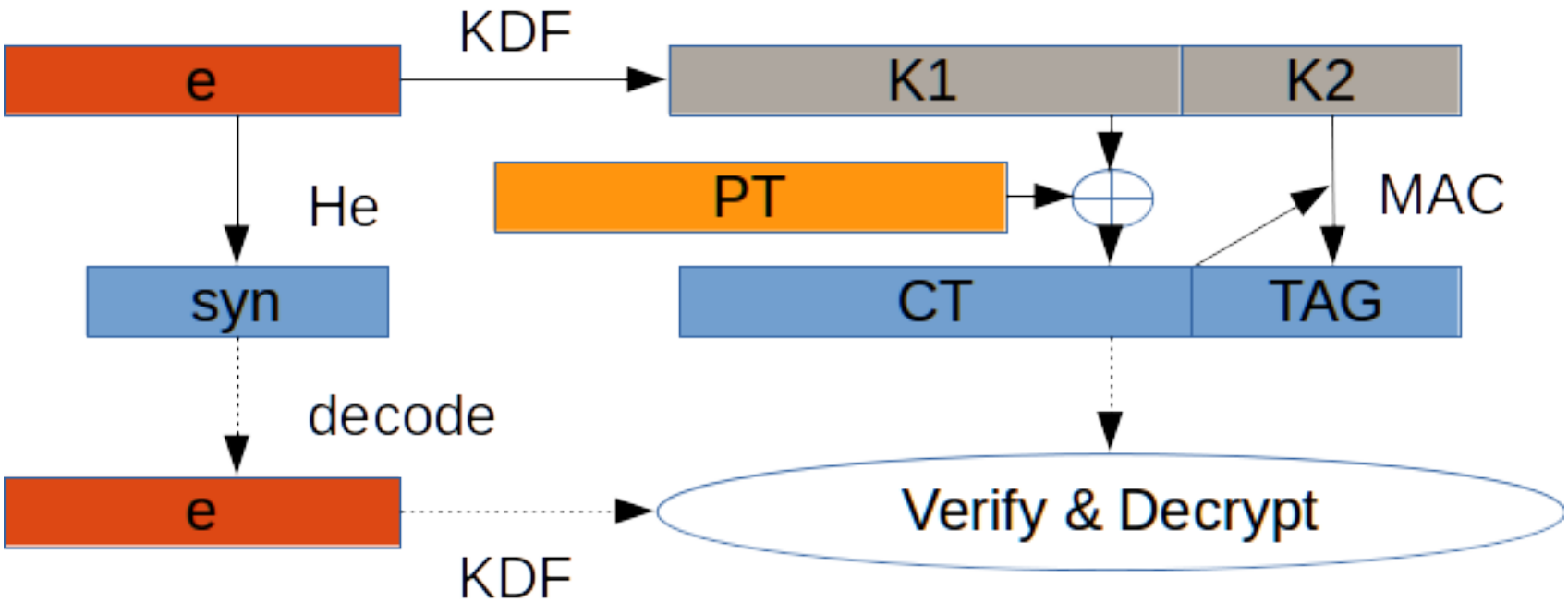


Kobara-Imai conversion — streaming version



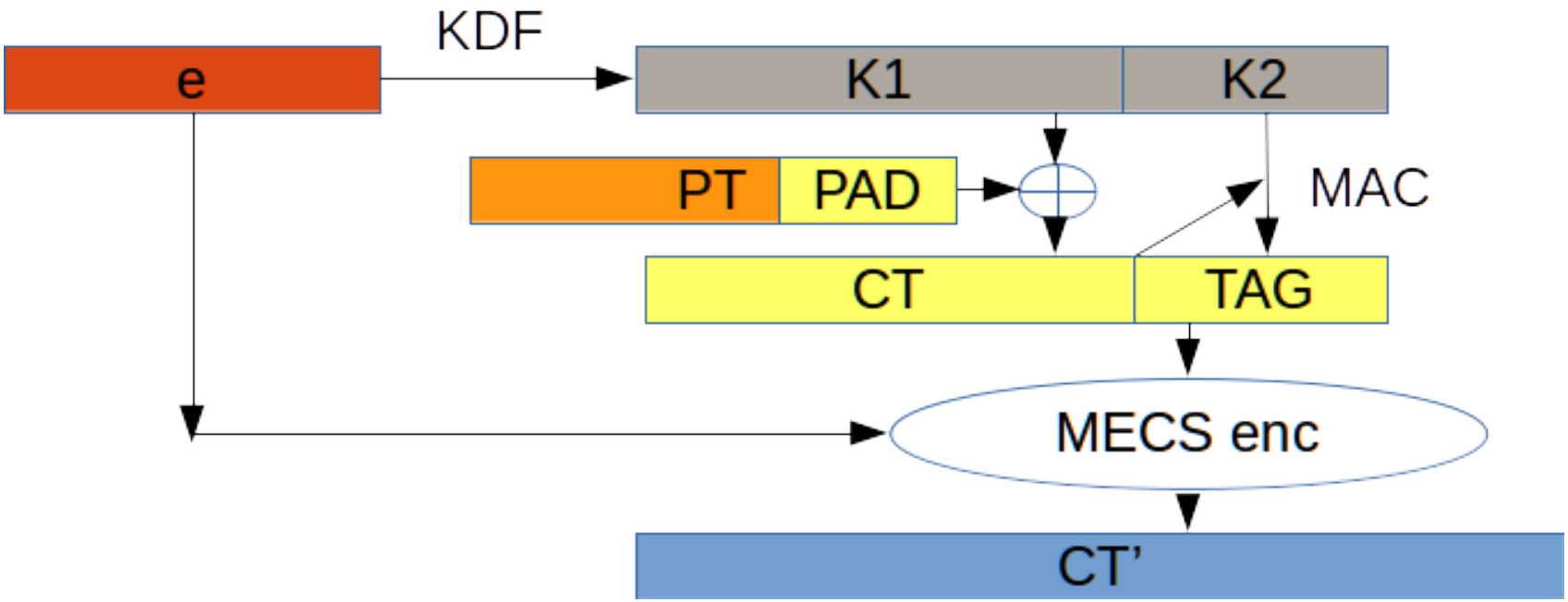
Hybrid encryption — KEM/DEM approach

Persichetti's Hybrid version of Niederreiter encryption:



Hybrid encryption — Cryptobox version

Our proposal based on MECS:



Hybrid schemes comparison

Persichetti (Niederreiter)	$n - k + len(PT) + len(MAC)$
Naive KEM/DEM McEliece	$n + len(PT) + len(MAC)$
Our KEM/DEM McEliece	$n - k + \max(len(PT) + len(MAC), k)$

Security:

1. (CT|TAG) behaves as random string,
2. (CT|TAG) can be verified only if e is found.
3. Reaction attacks: (CT|TAG) must be always verified, even if MECS decryption fails (potential side-channel).

Hybrid schemes comparison

Persichetti (Niederreiter)	$n - k + len(PT) + len(MAC)$
Naive KEM/DEM McEliece	$n + len(PT) + len(MAC)$
Our KEM/DEM McEliece	$n - k + \max(len(PT) + len(MAC), k)$

Security:

1. (CT|TAG) behaves as random string,
2. (CT|TAG) can be verified only if e is found.
3. Reaction attacks: (CT|TAG) must be always verified, even if MECS decryption fails (potential side-channel).

Hybrid schemes comparison

Persichetti (Niederreiter)	$n - k + len(PT) + len(MAC)$
Naive KEM/DEM McEliece	$n + len(PT) + len(MAC)$
Our KEM/DEM McEliece	$n - k + \max(len(PT) + len(MAC), k)$

Security:

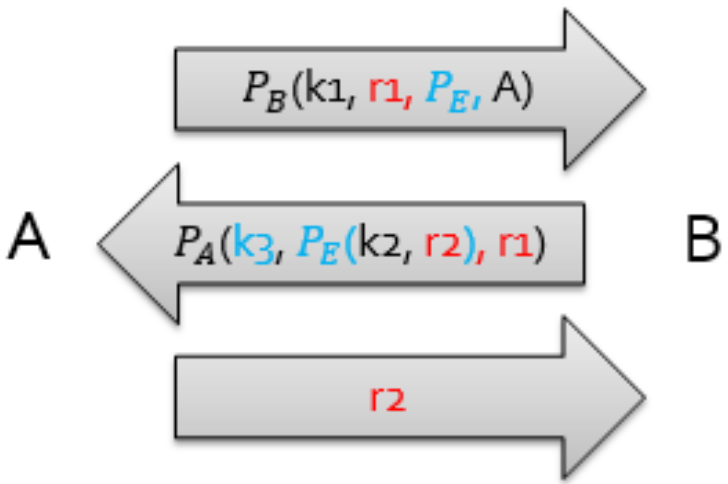
1. (CT|TAG) behaves as random string,
2. (CT|TAG) can be verified only if e is found.
3. Reaction attacks: (CT|TAG) must be always verified, even if MECS decryption fails (potential side-channel).

Android messenger application

- MSc. thesis: A. Boledovic
- Secure messenger for Android with McEliece encryption
  - Public Keys are handled by trusted server
  - McEliece encryption (+Kobara-Imai) is used to established symmetric session keys
  - Parameters: $m = 11, t = 50$ (140kB PK, 90+ bit sec.)
- Uses development version of BouncyCastle (Java)

Android messenger — protocol

Modified Needham-Schroeder with forward secrecy



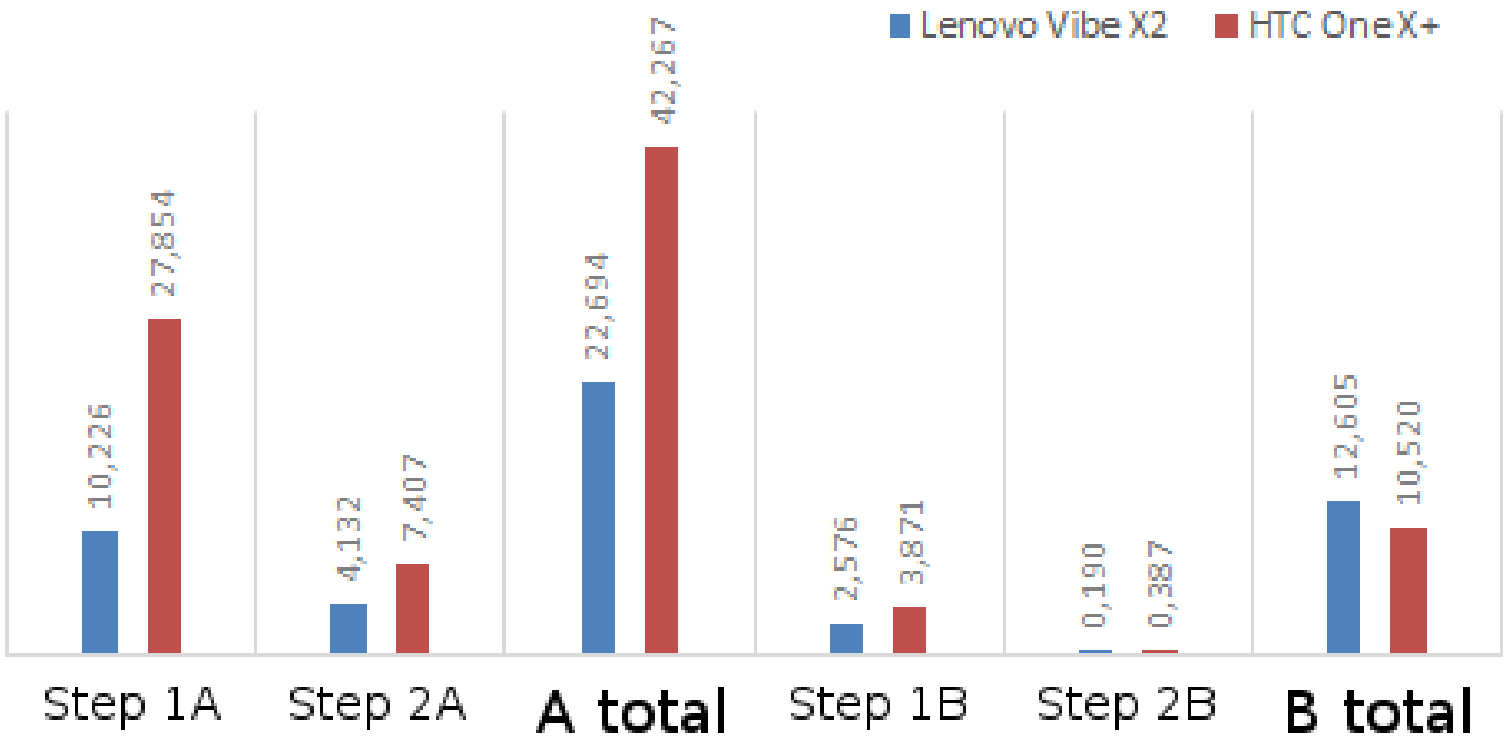
1A Generate ephemeral key + encrypt

2A 2× decrypt

1B 1× decrypt

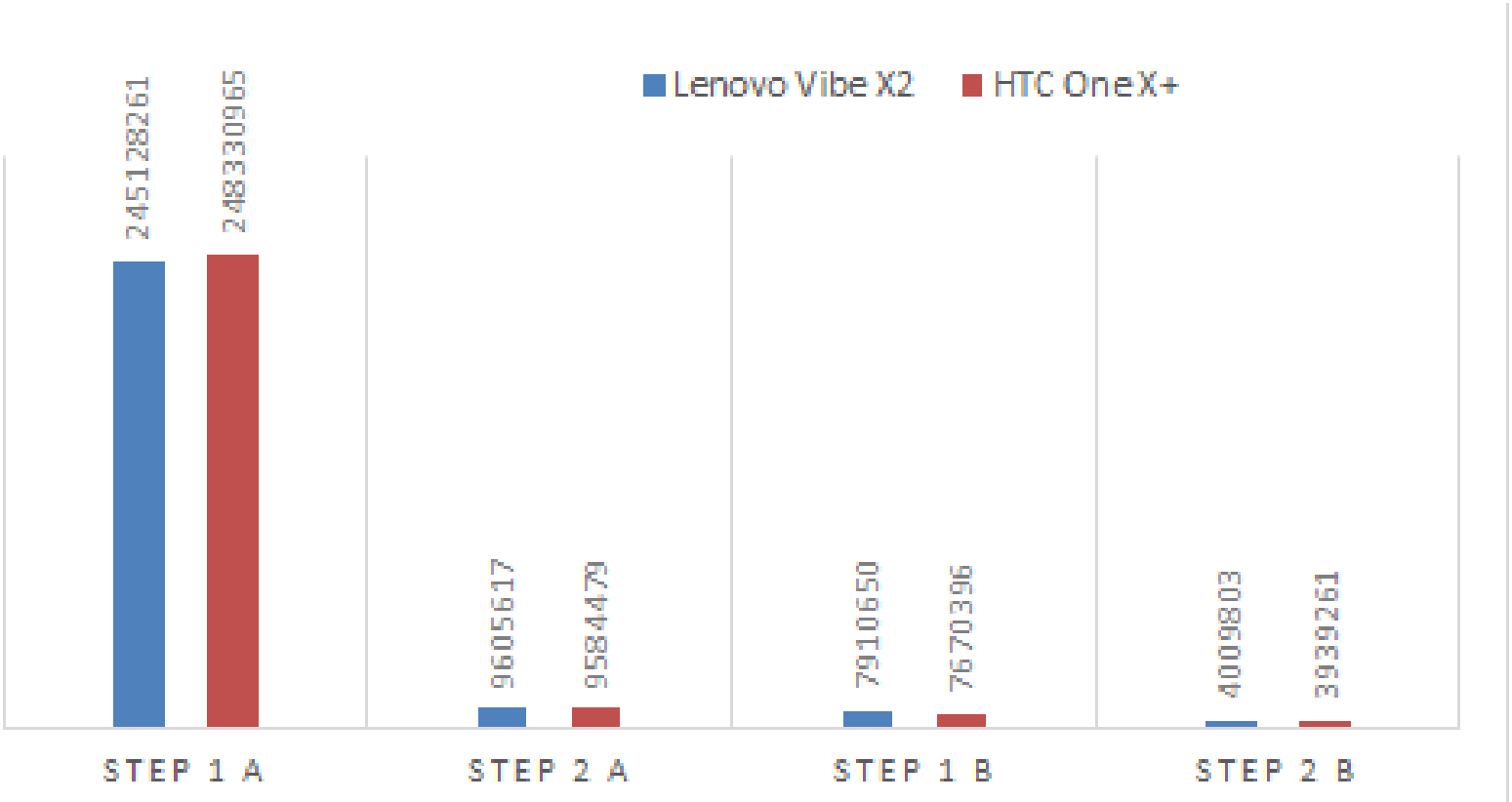
2B 2× encrypt

Android messenger — performance



Note: Total time includes communication overhead

Android messenger — memory

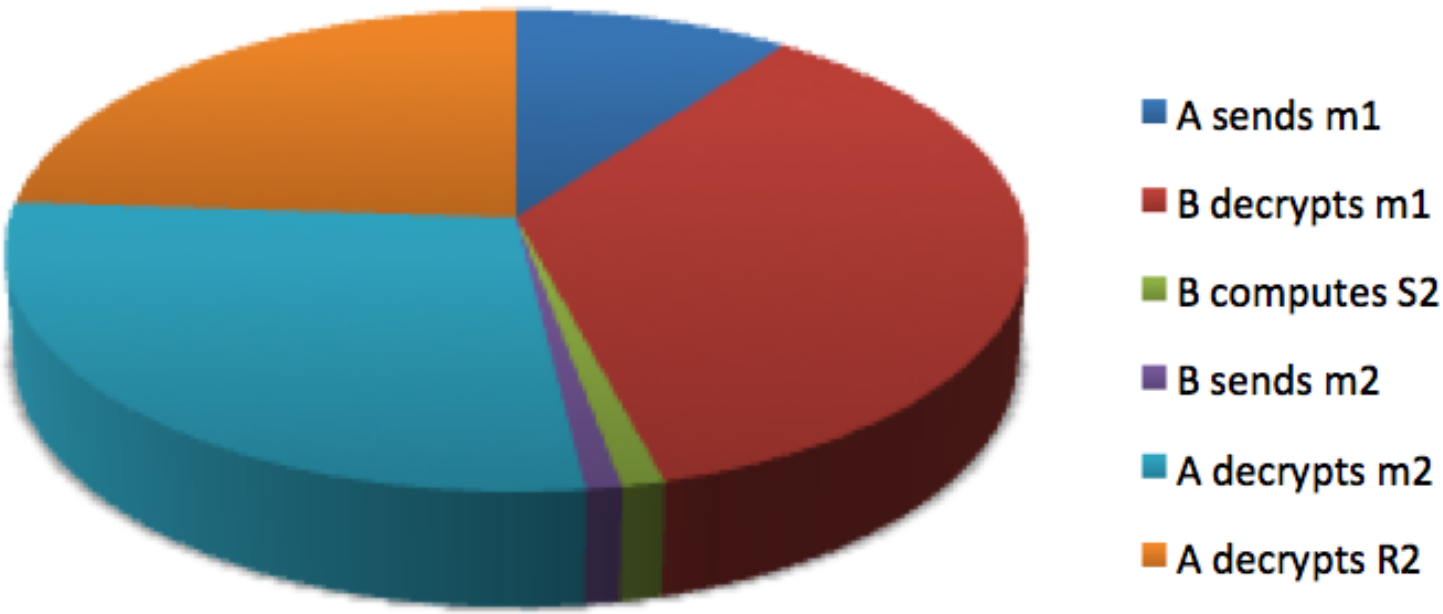


Note: Key generation in BC uses multiple large matrices

Forward-secrecy protocol with BitPunch

Ephemeral keygen: 1.5s

Enc/Dec: 0.050s



Summary

- Basic McEliece decryption must be secured against timing attacks.
- A practical CCA2 conversion/padding scheme is required:
  - Hybrid encryption / crypto-box approach seem preferable,
  - Potential side-channels / security problems with padding scheme.
- „Protocol balance”: Encryption is significantly faster.
- Forward secrecy: even if protocol overhead is acceptable, ephemeral key generation is too slow.

Questions, comments?