Solutions for the Storage Problem of McEliece Public and Private Keys on Memory-constrained Platforms

Falko Strenzke

FlexSecure GmbH, Darmstadt, Germany
strenzke@flexsecure.de

January 18, 2013

Introduction

- code-based cryptosystem built on error correcting codes
- McEliece, Niederreiter
- advantage: no efficient quantum algorithm known
- disadvantage: **key sizes**
- attempts to reduce public key size with "structured" codes
- **original** proposition of McEliece with Goppa Codes: **unbroken** for more than **30 years**

Goppa Codes

- Parameters of a Goppa Code
  - irreducible polynomial $g(Y) \in \mathbb{F}_{2^m}[Y]$ of degree t (the Goppa Polynomial)
  - support $\Gamma = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$, where α_i are pairwise distinct elements of $\mathbb{F}_{2^m}$
- Properties of the Code
  - the code has length $n \leq 2^m$ (code word length) ,
  - dimension $k = n - mt$ (message length) and
  - can correct up to t errors.
  - a parity check matrix $H \in \mathbb{F}_2^{mt \times n}$, where $cH^\top = 0$ if $c \in \mathcal{C}$
  - a generator matrix $G \in \mathbb{F}_2^{n \times k}$ with $\vec{m}G \in \mathcal{C}$
  - example for secure parameters: $n = 2048$, $t = 50$ for 102 bit security

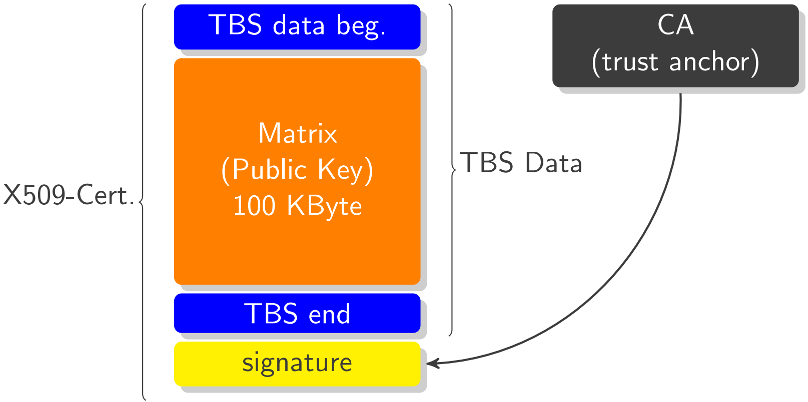
The McEliece PKC

- key generation
  - choose the parameters n and t
  - generate randomly $g(Y)$ and Γ (determining the secret the code)
  - for this private code $\mathcal{C}_s$ one has a public generator matrix G_s
  - the public key is $G_p = [\mathbb{I}|G_p'] = TG_s$
  - for 102 bit secure parameters: G_p' has size of about **100 KB**
- encryption: $\vec{z} = \vec{m}G_p + \vec{e}$, $\mathrm{wt}(\vec{e}) = t$
- decryption: knowing $g(Y)$ and Γ, $\vec{e}$ and thus also $\vec{m}$ can be recovered

Public Key Encryption

- McEliece is a public key encryption scheme
- i.e., applied in a Public Key Infrastructure (PKI) context

Encrpytion in PKI



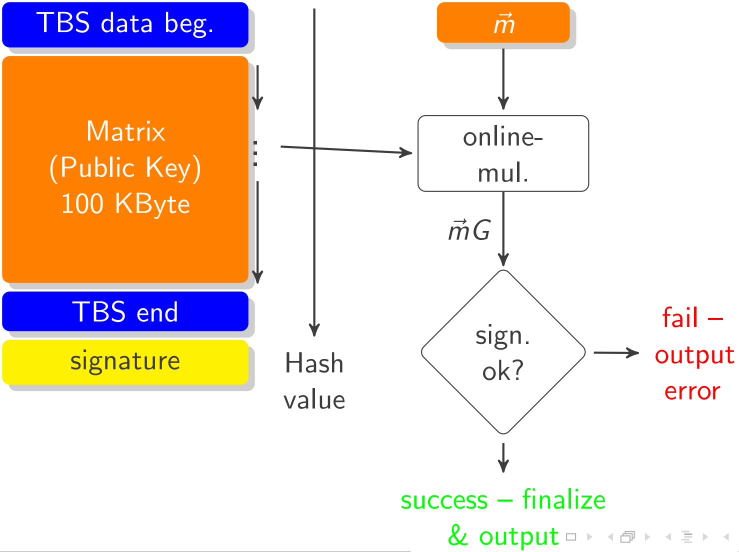
standard approach: transmitt the certificate, verify signature, encrypt with public key

Problems on Memory-constrained Platforms

- smart cards typically have less than 20 kB RAM
- $\rightarrow$ certificate/matrix in non-volatile memory
- $\rightarrow$ cost, slow writing speed, limited nr. write cylces
- why encryption on smart card?
- $\rightarrow$ in the context of electronic passports (Germany) and electronic health applications:
- key exchange schemes, can be built by signature schemes and PKCs

Solution for Memory-constrained Platforms

Process the certificate during receival:



Transmission Rates

- contactless smart card: up to 106 KByte/s (raw)
- transmit 100 KByte key (security $\approx$ 100 bit) in $\approx$1s
- research implementation by NXP Semiconductors 8 times faster
- $\rightarrow$ leaves 35 CPU cycles at 30MHz per byte

Computational Tasks

- SHA-256 Hash $\approx$ 30 cycles/byte on Pentium 4
- matrix multiplication column-wise:
  - AND of each column and $\vec{m}$ 32-bit word-wise
  - XOR result to 32-bit ACCU
  - finalize column: compute parity bit of ACCU

Example Implementation

- on Atmel AVR32 ATUC3A1512 32-bit microcontroller @ 33 MHz
- communicating with PC over RS232 @ 460,800 baud
- works with two interchanging buffers

Online-Multiplication Protocol

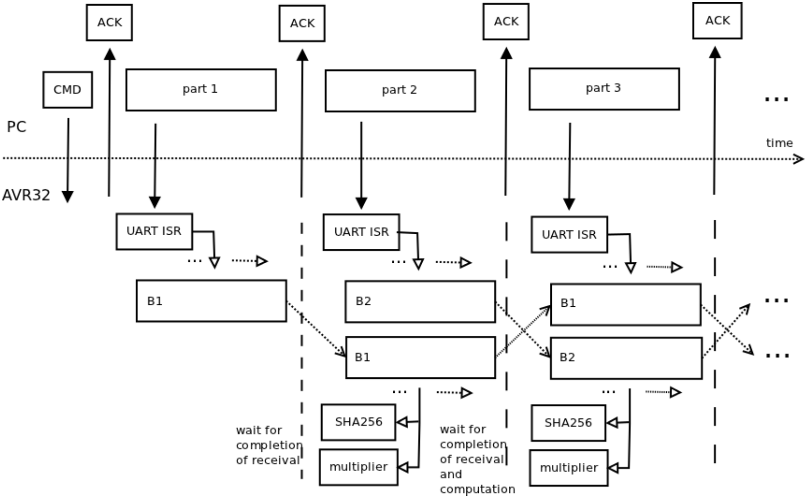


Figure: Schematic overview of the interrupt based implementation of the on-line multiplication.

Two Modifications to the Protocol

- non-interactive version
  - only the very first ACK is send
  - $\rightarrow$ faster by ≈ 1.3
- simulation of higher transmission speeds
  - use fake matrix with bytes repeating r times
  - i.e. `0x1D, 0x1D, 0x1D, 0x1D, 0xA3, 0xA3, 0xA3, 0xA3, 0x22, ...`
  - transmit repeated bytes only once
  - $B_{\mathrm{sim}} = rB_{\mathrm{real}}$

Results

	based on computation throughput	**experimental result - w/o ACK**
cycles/byte	measured: 55.6 for SHA-256, 4.2 for mult. yields: 59.8	92
time at 33MHz CPU for 100,000 Bytes	181ms	279ms
transmission rate in bytes/s	551,839	$B_{\text{sim}} = 368,640$ $(r = 8)$

- buffer size: 1536

- applicable basically all code-based schemes
  - McEliece PKC
  - Niederreiter PKC
  - CFS signature scheme
  - KKS signature scheme

Syndrome Computation with the Parity Check Matrix

- $S(Y) \in \mathbb{F}_{2^m}[Y]$ of degree $t-1$: starting point of decryption
- $\vec{s} = cH^T$
- interpret $\vec{s} \in \mathbb{F}_2^{mt}$ as coefficients ...
- $\rightarrow S(Y)$

McEliece Private Key Size

	size in bytes	
	$n = 2048$, $t = 50$, (102 bit)	$n = 2960$, $t = 56$ ($>$ 122 bit)
$4 \cdot 2^m$ bytes $\mathbb{F}_{2^m}$ tables	8,192	16,384
t^2 bytes table for square root in $\mathbb{F}_{2^m}[Y]/g(Y)$	2,500	3,136
$2t$ **bytes for** $g(Y)$	**100**	**112**
$2n$ **bytes for the support**	**4,048**	**5,920**
sum w/o Par. Ch. Mat.	14,840	25,552
Par. Ch. Mat.	140,800	248,640
sum w/Par. Ch. Mat.	155,640	274,192

Syndrome Computation without the Parity Check Matrix

- $S(Y) \equiv \sum_{i=1}^{n} \frac{c_i}{Y \oplus \alpha_i} \mod g(Y)$,
- where α_i is the i-th support element
- done with EEA in a single iteration
- EEA implementation can be optimized for this case

Optimized EEA

Require: the ciphertext $\vec{c} \in \mathbb{F}_2^n$, and the Goppa Polynomial $g(Y) \in \mathbb{F}_{2^m}[Y]$ of degree t
Ensure: the syndrome polynomial $S(Y) \in \mathbb{F}_{2^m}[Y]$ of degree $\leq t-1$

```
S(Y) ← 0
for i ← 0 up to n − 1 do
    if c⃗[i] = 1 then
        B(Y) ← 0
        b ← g_t
        for j ← t − 1 down to 0 do
            B_j ← b
            b ← b · α_i ⊕ g_j
        end for
        f ← b^{−1}
        for j ← 0 up to deg(B(Y)) do
            S_j ← S_j ⊕ f · B_j
        end for
    end if
end for
```

Cost of the Syndrome Computation

- $C_{\mathrm{syndr}} = nt(C_{\mathrm{mult}} + C_{\mathrm{add}}) + \frac{n}{2} C_{\mathrm{inv}}$
- an average
- except for the inversions: cost of root-finding with exhaustive search

Implementation

- platform: Atmel AT32 AP7000
- source code: HyMES Open Source McEliece C implementation https://www.rocq.inria.fr/secret/CBCrypto/index.php?pg=hymes

Experimental Results

code parameters		$n = 2048,\ t = 50$		$n = 2960,\ t = 56$	
security level		102 bit		$>$ 122 bit	

		cycles	t @ 33 MHz	cycles	t @ 33 MHz
with par. ch. mat.	whole decr.	$2.00 \cdot 10^6$	61 ms	$3.12 \cdot 10^6$	95 ms
	only syndr. comp.	$0.26 \cdot 10^6$	8 ms	$0.39 \cdot 10^6$	12 ms
	private key bytes	155,640		274,192	
w/o par. ch. mat.	whole decr.	$4.42 \cdot 10^6$	134 ms	$7.39 \cdot 10^6$	224 ms
	only synd. comp.	$2.65 \cdot 10^6$	80 ms	$4,71 \cdot 10^6$	143 ms
	private key bytes	14,840		25,552	

Conclusion

- code-based public operations in a PKI context: transmission speed is the limiting factor
- applicability in certain scenarios seems possible even today
- syndrome computation without the parity check matrix is still efficient
- $\rightarrow$ advantage of McEliece over Niederreiter

Thank you!

download the McEliece implementation and these slides:
http://crypto-source.de