Post-Quantum Cryptography

Tanja Lange

Technische Universiteit Eindhoven

tanja@hyperelliptic.org

09 November 2009

Threat of quantum computers

Shor's algorithm makes polynomial time:

- integer factorization
- DLP in finite fields
- DLP on elliptic curves
- DLP in general class groups

Grover's algorithm brings faster simultaneous search in data

- some security loss in symmetric crypto (block and stream ciphers)
- some security loss in hash functions (if not VSH)

Compensate for Grover by $\approx$ doubling key size.

Threat of quantum computers

Shor's algorithm makes polynomial time:

- integer factorization
- DLP in finite fields
- DLP on elliptic curves
- DLP in general class groups

Grover's algorithm brings faster simultaneous search in data

- some security loss in symmetric crypto (block and stream ciphers)
- some security loss in hash functions (if not VSH)

Compensate for Grover by $\approx$ doubling key size.

Sorry,
no picture
available

… but 15 years from now …

Large quantum computers might be reality. Then

- RSA is dead.
- DH key exchange is dead.
- DSA is dead.
- XTR is dead.
- ECDSA is dead.
- ECC is dead.
- HECC is dead.

…but 15 years from now …

Large quantum computers might be reality. Then

- RSA is dead.
- DH key exchange is dead.
- DSA is dead.
- XTR is dead.
- ECDSA is dead.
- ECC is dead.
- HECC is dead.
- all public key cryptography

. . . but 15 years from now . . .

Large quantum computers might be reality. Then

- RSA is dead.
- DH key exchange is dead.
- DSA is dead.
- XTR is dead.
- ECDSA is dead.
- ECC is dead.
- HECC is dead.
- all public key cryptography is dead?

. . . but 15 years from now . . .

Large quantum computers might be reality. Then

- RSA is dead.
- DH key exchange is dead.
- DSA is dead.
- XTR is dead.
- ECDSA is dead.
- ECC is dead.
- HECC is dead.
- all public key cryptography is dead?
- Actually there are a few more public-key cryptosystems.

The “survivors”

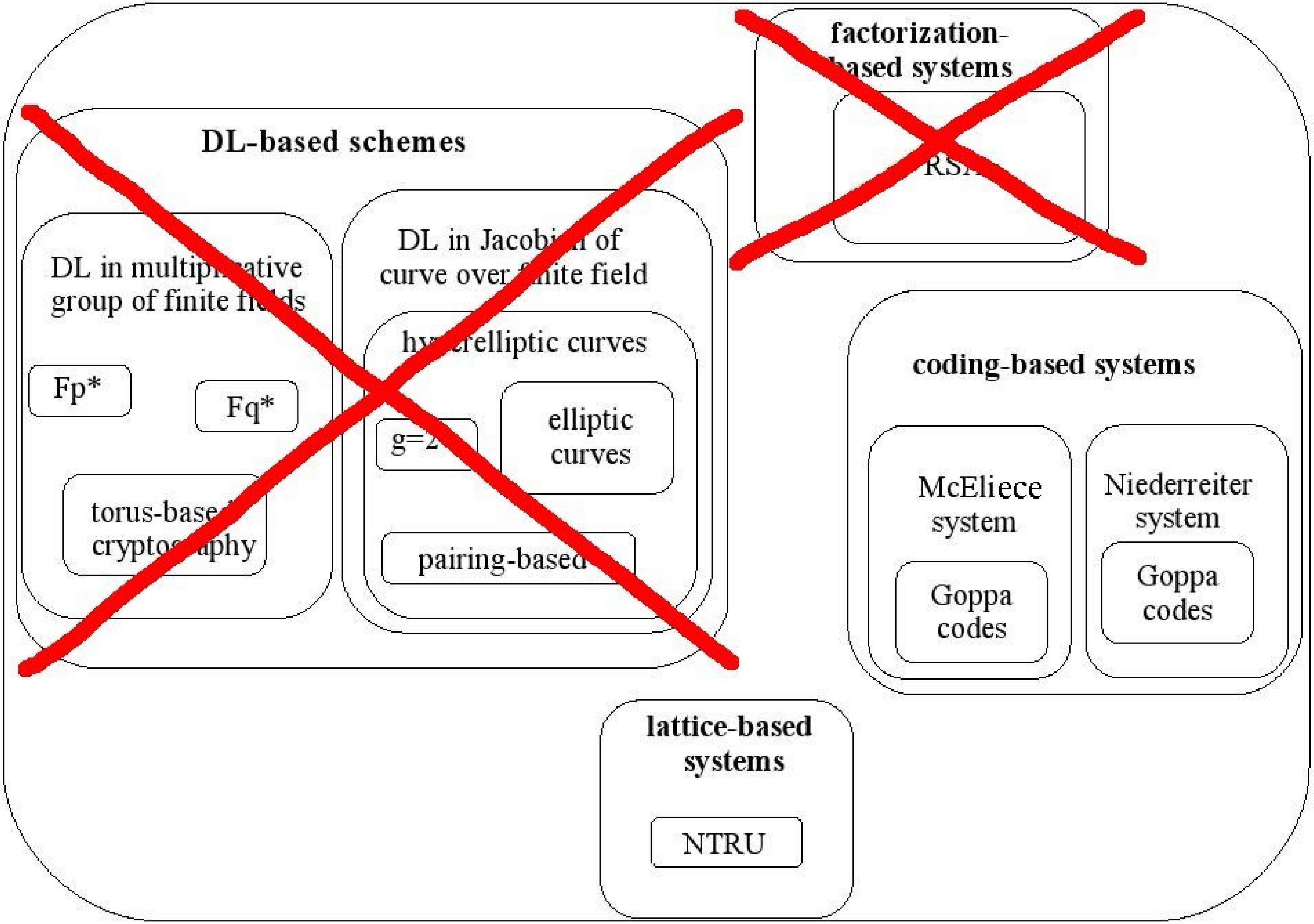
Public-key encryption:

- Lattice-based cryptography (e.g. NTRU)
- Code-based cryptography (e.g. McEliece, Niederreiter)

Public-key signatures:

- Multivariate-quadratic-equations cryptography (e.g. HFE^{-})
- Hash based cryptography (e.g. Merkle’s hash-trees signatures)

For these systems no efficient usage of Shor’s algorithm is known. Grover’s algorithm has to be taken into account when choosing key sizes.
Some more possibilities with less confidence.



Encryption systems:

Why care about this now?

15 years might seem a long time. But

- There is no guarantee that it takes at least 15 years.
- Long-term confidential documents (e.g. health records, state secrets) become readable once quantum computers are available.
- Electronic signatures on long-term commitments (e.g. last wishes, contracts) can be forged once quantum computers are available.
- Nobody will inform you if a secret agency made a breakthrough in constructing a quantum computer.
- The systems mentioned before remain secure – but need better study of parameters, faster implementations, and better embedding into protocols.

How about quantum cryptography?

- Quantum cryptography expands a short shared key into an effectively infinite shared stream.
- Requires Alice and Bob to know some (e.g. 256) unpredictable secret key bits.
- Result of quantum cryptography is that Alice and Bob both know a stream of some more (e.g. 10^{12}) unpredictable secret bits.
- Length of the output stream increases linearly with the amount of time.

How about quantum cryptography?

- Quantum cryptography expands a short shared key into an effectively infinite shared stream.
- Requires Alice and Bob to know some (e.g. 256) unpredictable secret key bits.
- Result of quantum cryptography is that Alice and Bob both know a stream of some more (e.g. 10^{12}) unpredictable secret bits.
- Length of the output stream increases linearly with the amount of time.
- Sounds like a stream cipher to you? Not exactly . . .

Differences from stream ciphers

- Quantum cryptography uses physical techniques instead of mathematical function of the input key.
- Quantum cryptography needs direct connection between the quantum cryptography hardware (distance is an issue), eavesdropping interrupts the communication. Conventional cryptography can use standard channels; eavesdropping fails because the encrypted information is incomprehensible.
- Security of quantum cryptography follows from quantum mechanics instead of being merely conjectural.

Differences from stream ciphers

- Quantum cryptography uses physical techniques instead of mathematical function of the input key.
- Quantum cryptography needs direct connection between the quantum cryptography hardware (distance is an issue), eavesdropping interrupts the communication. Conventional cryptography can use standard channels; eavesdropping fails because the encrypted information is incomprehensible.
- Security of quantum cryptography follows from quantum mechanics instead of being merely conjectural.
- A stream cipher can be implemented on conventional CPUs and generates GB of stream per second on a \$200 CPU. Quantum cryptography generates kB of stream per second on special hardware costing \$50000.

How about quantum cryptography?

- Quantum cryptography expands a short shared key into an effectively infinite shared stream.
- Requires Alice and Bob to know some (e.g. 256) unpredictable secret key bits.
- Result of quantum cryptography is that Alice and Bob both know a stream of some more (e.g. 10^{12}) unpredictable secret bits.
- Length of the output stream increases linearly with the amount of time.

How about quantum cryptography?

- Quantum cryptography expands a short shared key into an effectively infinite shared stream.
- Requires Alice and Bob to know some (e.g. 256) unpredictable secret key bits.
- Result of quantum cryptography is that Alice and Bob both know a stream of some more (e.g. 10^{12}) unpredictable secret bits.
- Length of the output stream increases linearly with the amount of time.
- More serious problem: how to get the initial secret?! Secret meeting of agents and key exchange – or public-key cryptography.

How about quantum cryptography?

- Quantum cryptography expands a short shared key into an effectively infinite shared stream.
- Requires Alice and Bob to know some (e.g. 256) unpredictable secret key bits.
- Result of quantum cryptography is that Alice and Bob both know a stream of some more (e.g. 10^{12}) unpredictable secret bits.
- Length of the output stream increases linearly with the amount of time.
- More serious problem: how to get the initial secret?! Secret meeting of agents and key exchange – or public-key cryptography.
- And there was no problem in symmetric cryptography in the first place.

Post-quantum cryptography

- Cryptographic systems that run on conventional computers, are secure against attacks with conventional computers, and remain secure under attacks with quantum computers are called post-quantum cryptosystems.
- Post-quantum cryptography deals with
  - the design of such systems;
  - cryptanalysis of such systems;
  - the analysis of suitable parameters depending on different threat models;
  - design of protocols using the secure primitives.
- Could potentially develop cryptosystems for quantum computers – but there won't be a market too soon.

Warnings

- The following describes text-book versions.
- There exist e.g. CCA2 secure versions, versions with better efficiency, other finite fields . . . .

Hash-based signatures

- Evolved from Lamport one-time signature scheme.
- Standardize a $4k$-bit hash function H.
- Signer's secret key X: $8k$ strings $x_1[0], x_1[1], \ldots, x_{4k}[0], x_{4k}[1]$, each $4k$ bits. Total: $32k^2$ bits.
- Signer's public key Y: $8k$ strings $y_1[0], y_1[1], \ldots, y_{4k}[0], y_{4k}[1]$, each $4k$ bits, computed as $y_i[b] = H(x_i[b])$. Total: $32k^2$ bits.
- Signature $S(X, r, m)$ of a message m: $r, x_1[h_1], \ldots, x_{4k}[h_{4k}]$ where $H(r, m) = (h_1, \ldots, h_{4k})$.
- Must never use secret key more than once.
- Collision resistance not required from H.
- 1979 Merkle extends to more signatures.

4-time Merkle hash tree

Four Lamport one-time keys Y_1, Y_2, Y_3, Y_4 with corresponding X_1, X_2, X_3, X_4

$$\begin{tikzcd}[column sep=small]
X_1 \arrow[d] & & X_2 \arrow[d] & & X_3 \arrow[d] & & X_4 \arrow[d] \\
Y_1 \arrow[dr] & & Y_2 \arrow[dl] & & Y_3 \arrow[dr] & & Y_4 \arrow[dl] \\
& Y_5 = H(Y_1, Y_2) \arrow[drr] & & & & Y_6 = H(Y_3, Y_4) \arrow[dll] & \\
& & & Y_7 = H(Y_5, Y_6) & & &
\end{tikzcd}$$

Merkle public key is Y_7.

4-time Merkle hash tree

First message m_1: signature $S(X_1, r_1, m_1), Y_1, Y_2, Y_5, Y_6$.

$$\begin{tikzcd}
X_1 \arrow[d] & & X_2 \arrow[d] & & X_3 \arrow[d] & & X_4 \arrow[d] \\
Y_1 \arrow[dr] & & Y_2 \arrow[dl] & & Y_3 \arrow[dr] & & Y_4 \arrow[dl] \\
& Y_5 = H(Y_1, Y_2) \arrow[drr] & & & & Y_6 = H(Y_3, Y_4) \arrow[dll] & \\
& & & Y_7 = H(Y_5, Y_6) & & &
\end{tikzcd}$$

Receiver's verification:
$Y_7 = H(Y_5, Y_6)$;
$Y_5 = H(Y_1, Y_2)$;
Y_1 matches one-time signature $S(X_1, r_1, m_1)$.

4-time Merkle hash tree

Second message m_2: signature $S(X_2, r_2, m_2), Y_2, Y_1, Y_5, Y_6$.

$$
\begin{tikzcd}
X_1 \arrow[d] & & X_2 \arrow[d] & & X_3 \arrow[d] & & X_4 \arrow[d] \\
Y_1 \arrow[dr] & & Y_2 \arrow[dl] & & Y_3 \arrow[dr] & & Y_4 \arrow[dl] \\
& Y_5 = H(Y_1, Y_2) \arrow[drr] & & & & Y_6 = H(Y_3, Y_4) \arrow[dll] & \\
& & & Y_7 = H(Y_5, Y_6) & & &
\end{tikzcd}
$$

Receiver's verification:
$Y_7 = H(Y_5, Y_6)$;
$Y_5 = H(Y_1, Y_2)$ (after trying $H(Y_2, Y_1)$);
Y_2 matches one-time signature $S(X_2, r_2, m_2)$.

4-time Merkle hash tree

Third message m_3: signature $S(X_3, r_3, m_3), Y_3, Y_4, Y_6, Y_5$.

$$
\begin{tikzcd}[column sep=small]
X_1 \arrow[d] & & X_2 \arrow[d] & & X_3 \arrow[d] & & X_4 \arrow[d] \\
Y_1 \arrow[dr] & & Y_2 \arrow[dl] & & Y_3 \arrow[dr] & & Y_4 \arrow[dl] \\
& Y_5 = H(Y_1, Y_2) \arrow[drr] & & & & Y_6 = H(Y_3, Y_4) \arrow[dll] & \\
& & & Y_7 = H(Y_5, Y_6) & & &
\end{tikzcd}
$$

Receiver's verification:
$Y_7 = H(Y_5, Y_6)$;
$Y_6 = H(Y_3, Y_4)$;
Y_3 matches one-time signature $S(X_3, r_3, m_3)$.

4-time Merkle hash tree

Fourth message m_4: signature $S(X_4, r_4, m_4), Y_4, Y_3, Y_6, Y_5$.

$$
\begin{tikzcd}
X_1 \arrow[d] & & X_2 \arrow[d] & & X_3 \arrow[d] & & X_4 \arrow[d] \\
Y_1 \arrow[dr] & & Y_2 \arrow[dl] & & Y_3 \arrow[dr] & & Y_4 \arrow[dl] \\
& Y_5 = H(Y_1, Y_2) \arrow[dr] & & & & Y_6 = H(Y_3, Y_4) \arrow[dl] & \\
& & & Y_7 = H(Y_5, Y_6) & & &
\end{tikzcd}
$$

Receiver's verification:
$Y_7 = H(Y_5, Y_6)$;
$Y_6 = H(Y_3, Y_4)$ (after trying $H(Y_4, Y_13)$);
Y_4 matches one-time signature $S(X_4, r_4, m_4)$.

Multivariate signatures

- Idea: Given $y_0, \ldots, y_{n-1} \in \mathbb{F}_2$ finding $x_0, \ldots, x_{n-1} \in \mathbb{F}_2$ with
$$\begin{aligned} y_0 &= q_0(x_0, x_1, \ldots, x_{n-1}), \\ y_1 &= q_1(x_0, x_1, \ldots, x_{n-1}), \\ &\vdots \\ y_{n-1} &= q_{n-1}(x_0, x_1, \ldots, x_{n-1}), \end{aligned}$$
is hard, where the q_i are quadratic equation over $\mathbb{F}_2$.
- Signature: preimage of $(y_0, \ldots, y_{n-1}) = H(r, m)$ (if exists).
- Build in trapdoor by constructing the polynomials from a hidden polynomial $q(x)$ over $\mathbb{F}_{2^n} \cong \mathbb{F}_2[t]/f(t)$, using $x = \sum x_i t^i$ and sorting by powers of t. Finding $x \in \mathbb{F}_{2^n}$ with $q(x) = y$ easier.
- Hide structure by applying linear transformations, removing some equations; adding extra variables.

Attacking and Defending the McEliece Cryptosystem

joint work with

Daniel J. Bernstein and Christiane Peters

Thanks to Christiane for several slides.

Linear codes

We only consider binary codes, i.e. codes over $\mathbb{F}_2$.

- A generator matrix of an $[n, k]$ code C is a $k \times n$ matrix G such that $C = \{\mathbf{x}\, G : \mathbf{x} \in \mathbb{F}_2^k\}$.
- The matrix G corresponds to a map $\mathbb{F}_2^k \rightarrow \mathbb{F}_2^n$ sending a message of length k to an n-bit string.
- A parity-check matrix of an $[n, k]$ code C is an $(n-k) \times n$ matrix H such that $C = \{\mathbf{c} \in \mathbb{F}_2^n : H\, \mathbf{c}^T = 0\}$.
- A systematic generator matrix is a generator matrix of the form $(I_k|Q)$ where I_k is the $k \times k$ identity matrix and Q is a $k \times (n-k)$ matrix (redundant part).
- Easy to get parity-check matrix from systematic generator matrix, use $H = (Q^T|I_{n-k})$.

Decoding problem

- The Hamming distance between two words in $\mathbb{F}_2^n$ is the number of coordinates where they differ. The Hamming weight of a word is the number of non-zero coordinates.
- The minimum distance of a linear code C is the smallest Hamming weight of a nonzero codeword in C.
- Classical decoding problem: find the closest codeword $\mathbf{x} \in C$ to a given $\mathbf{y} \in \mathbb{F}_2^n$, assuming that there is a unique closest codeword.
- In particular: Decoding a generic binary code of length n and without knowing anything about its structure requires about $2^{(0.5+o(1))n/\log_2(n)}$ binary operations (assuming a rate $\approx 1/2$)
- Coding theory deals with efficiently decodeable codes.

The McEliece cryptosystem I

- Let C be a length-n binary Goppa code Γ of dimension k with minimum distance $2t + 1$ where $t \approx (n - k)/\log_2(n)$; (original parameters: $n = 1024$, $k = 524$, $t = 50$).
- The McEliece secret key consists of a generator matrix G for Γ, an efficient t-error correcting decoding algorithm for Γ; an $n \times n$ permutation matrix P and a nonsingular $k \times k$ matrix S.
- n, k, t are public; but Γ, P, S are randomly generated secrets.
- The McEliece public key is the $k \times n$ matrix $G' = SGP$.

The McEliece cryptosystem II

- McEliece encryption: Compute $\mathbf{m}G'$ and add a random error vector $\mathbf{e}$ of weight t and length n.
- Encryption of a message $\mathbf{m}$ of length k: Compute $\mathbf{m}G'$ and add a random error vector $\mathbf{e}$ of weight t and length n. Send $\mathbf{y} = \mathbf{m}G' + \mathbf{e}$.
- McEliece decryption using secret key: Compute $\mathbf{y}P^{-1} = \mathbf{m}G'P^{-1} + \mathbf{e}P^{-1} = \mathbf{m}SG + \mathbf{e}P^{-1}$.
  Use decoding algorithm to find $\mathbf{m}S$ and thereby $\mathbf{m}$.
- Attacker is faced with decoding $\mathbf{y}$ to nearest codeword $\mathbf{m}G'$ in the code generated by G'. This is general decoding if G' does not expose any structure.
- For codes other than Goppa codes often original code could be reconstructed from G' allowing faster decoding.

Attacks on the McEliece PKC

- Most effective attack against the McEliece cryptosystem (with binary Goppa code) is information-set decoding.
- Many variants:
  McEliece (1978), Leon (1988), Lee and Brickell (1988), Stern (1989), van Tilburg (1990), Canteaut and Chabanne (1994), Canteaut and Chabaud (1998), and Canteaut and Sendrier (1998).
- Note: Our complexity analysis showed that Stern's original attack beats Canteaut et al. when aiming for 128-bit security.
- Our attack is most easily understood as a variant of Stern's attack.

Search for low weight words vs. decoding

- McEliece ciphertext $\mathbf{y} \in \mathbb{F}_2^n$ has distance t from a unique closest codeword $\mathbf{c} = \mathbf{m}G$ in a code C which has minimum distance at least $2t + 1$.
- Find $\mathbf{e}$ of weight t such that $\mathbf{c} = \mathbf{y} - \mathbf{e}$:
  - append $\mathbf{y}$ to the list of generators
  - and form a generator matrix for $C + \{0, \mathbf{y}\}$.

  Then

  $$\mathbf{e} = (\mathbf{m}, 1) \left(\frac{G}{\mathbf{m}G + \mathbf{e}} \right)$$

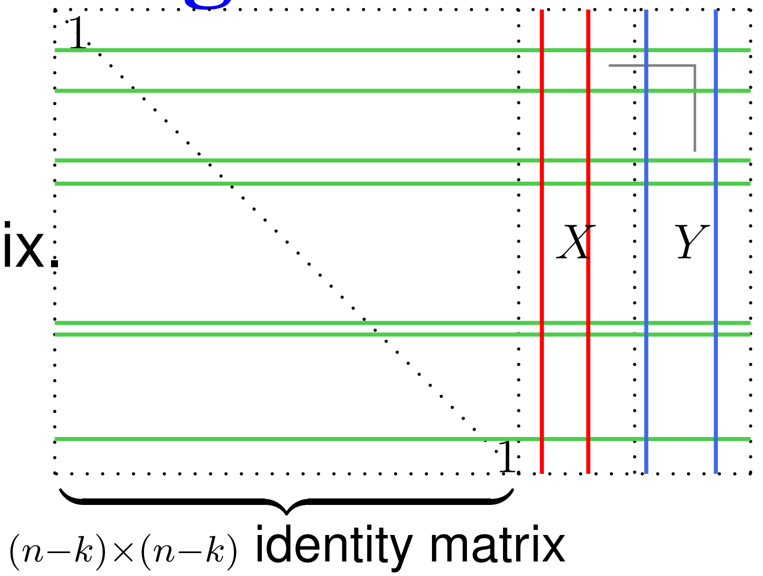
  is a codeword in $C + \{0, \mathbf{y}\}$; and it is the only weight-t word.
- Bottleneck in all of these attacks is finding the weight-t codeword in $C + \{0, \mathbf{y}\}$. This code has dimension $k + 1$.

Stern's attack

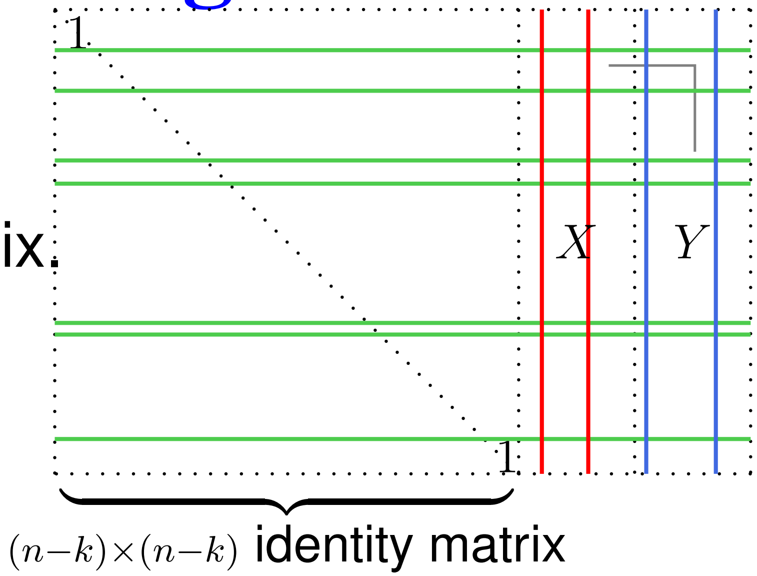
- Given $w \geq 0$ and an $(n-k) \times n$ parity check matrix H for a binary $[n,k]$ code C. Find $\mathbf{c} \in C$ of weight w.
- Construct $\mathbf{c}$ by looking for exactly w columns of H which add up to 0.
- Stern: Choose three disjoint subsets X, Y, Z among the columns of H.
  Search for words having exactly $p, p, 0$ ones in those column sets and exactly $w - 2p$ ones in the remaining columns.

One iteration of Stern's algorithm

- Select $n-k$ linearly independent columns; apply elementary row operations to get the identity matrix.
- Form a set Z of ℓ rows.
- Divide remaining k columns into two subsets X and Y.

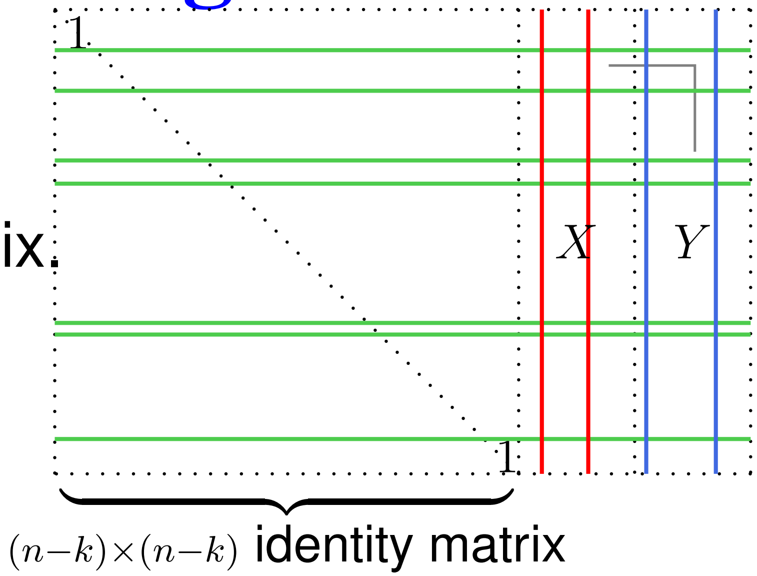


One iteration of Stern's algorithm



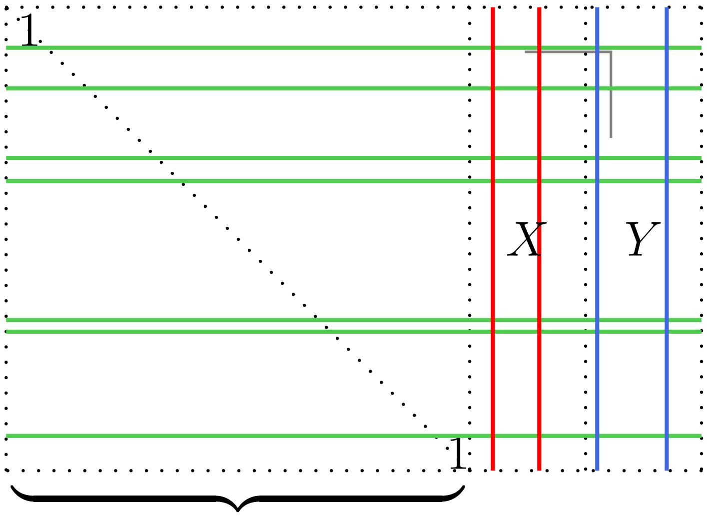
- Select $n-k$ linearly independent columns; apply elementary row operations to get the identity matrix.
- Form a set Z of ℓ rows.
- Divide remaining k columns into two subsets X and Y.
- For every size-p subset A of X compute the ℓ-bit vector $\pi(A)$ by adding up the columns of $H' = (H_{i,j})_{i\in Z, j\in A}$. Similarly, compute $\pi(B)$.
- For each collision $\pi(A) = \pi(B)$ compute the sum of the $2p$ columns in $A \cup B$. This sum is an $(n-k)$-bit vector.

One iteration of Stern's algorithm



- Select $n-k$ linearly independent columns; apply elementary row operations to get the identity matrix.
- Form a set Z of ℓ rows.
- Divide remaining k columns into two subsets X and Y.
- For every size-p subset A of X compute the ℓ-bit vector $\pi(A)$ by adding up the columns of $H' = (H_{i,j})_{i\in Z, j\in A}$. Similarly, compute $\pi(B)$.
- For each collision $\pi(A) = \pi(B)$ compute the sum of the $2p$ columns in $A \cup B$. This sum is an $(n-k)$-bit vector.
- **If** the sum has weight $w-2p$, we obtain 0 by adding the corresponding $w-2p$ columns in the $(n-k)\times(n-k)$ submatrix. **Else** select $n-k$ new columns.

Our improvements



Step 1

- Starting linear algebra part by using previous iteration's column selection.
- Forcing more existing pivots: reuse exactly $n-k-c$ column selections (Canteaut et al.: $c=1$).
- Faster pivoting.
- Multiple choices of Z: allow m disjoint sets $Z_1, \ldots, Z_m$ s.t. the word we're looking for has weight $(p, p, 0)$ on the set (X, Y, Z_i) for at least one $1 \leq i \leq m$.

Step 2

- Reusing additions of the ℓ-bit vectors for p-element subsets A of X.
- Faster additions after collisions: consider at most w instead of $n-k$ columns.

Bounding the number of iterations

- Stern: iterations are independent (in each step $n-k$ linearly independent columns are randomly chosen).
- Our attack reuses existing pivots: Number of errors in the selected $n-k$ columns is correlated with the number of errors in the columns selected in the next iteration.
- Extreme case $c=1$ considered by Canteaut et al.: swapping one selected column for one deselected column is quite likely to preserve the number of errors in the selected columns.
- We analyzed the impact of selecting c new columns on the number of iterations with a Markov chain computation (generalizing from Canteaut et al.).
  www.win.tue.nl/~cpeters/mceliece.html

Complexity of attacking original McEliece

- Canteaut, Chabaud, and Sendrier: an attacker can decode 50 errors in a $[1024, 524]$ code over $\mathbb{F}_2$ in $2^{64.1}$ bit operations.
- Choosing parameters $p = 2$, $m = 2$, $\ell = 20$, $c = 7$, and $r = 7$ in our new attack shows that the same computation can be done in only $2^{60.55}$ bit operations, almost a $12\times$ improvement over Canteaut et al.
- The number of iterations drops from $9.85 \cdot 10^{11}$ to $4.21 \cdot 10^{11}$, and the number of bit operations per iteration drops from $20 \cdot 10^6$ to $4 \cdot 10^6$.

Running time in practice

- Wrote attack software against original McEliece parameters, decoding 50 errors in a $[1024, 524]$ code.
- Attack on a single computer with a 2.4GHz Intel Core 2 Quad Q6600 CPU would need, on average, 1400 days (2^{58} CPU cycles) to complete the attack.
- Running the software on 200 such computers would reduce the average time to one week.
- Canteaut, Chabaud, and Sendrier: implementation on a 433MHz DEC Alpha CPU; one such computer would need approximately 7400000 days (2^{68} CPU cycles).
- Note: Hardware improvements only reduce 7400000 days to 220000 days (Moore's law).
- The remaining speedup factor of 150 comes from our improvements of the attack itself.

First successful attack

- About 200 computers involved, with about 300 cores.
- Computation finished in under 90 days (starting in July, ending at the beginning of October).
- Most of the cores put in far fewer than 90 days of work; some of which were considerably slower than a Core 2.
- Computation used about 8000 core-days.
- Error vector found by Walton cluster at SFI/HEA Irish Centre of High-End Computing (ICHEC).
- Tuned attack parameters after start of the computation. Later computers started with $m = 2$, $c = 12$.
- Using the new parameters the whole computation should take only 5000 core-days on average.

Contributed CPU cycles from

- the Coding and Cryptography Computer Cluster (C4) at TU/e;
- the FACS cluster at CWI;
- the Walton cluster at SFI/HEA Irish Centre for High-End Computing (ICHEC);
- the Department of Electrical Engineering at National Taiwan University;
- the CACAO cluster at LORIA;
- the sandpit Cluster at TU/e;
- the Argo cluster;
- the Center for Research and Instruction in Technologies for Electronic Security (RITES) at UIC;
- and D. J. Bernstein and Tanja Lange.

Advertisement

- Visit www.pqcrypto.org for much more material, in particular references in
  - Quantum computing
  - Hash-based cryptography
  - Code-based cryptography
  - Lattice-based cryptography
  - MQ cryptography
- Help us complete the bibliography.
- Next conference on post-quantum cryptography: PQCrypto 2010 in Darmstadt, Germany, May 25-28. Dates directly before Eurocrypt (Nice, France). Location close to Frankfurt airport.

Improvements

- Increasing n
  The most obvious way to defend McEliece's cryptosystem is to increase the code length n.
- Allowing values of n between powers of 2 allows considerably better optimization of (e.g.) the McEliece public-key size.
- Using list decoding to increase w
  2008: Bernstein introduced a list-decoding algorithm for classical irreducible binary Goppa codes:
  The receiver can efficiently decode approximately $n - \sqrt{n(n-2t-2)} \geq t+1$ errors instead of t errors.
  The sender can introduce correspondingly more errors.
  Unique decoding is ensured by CCA2-secure variants.

New proposed parameters

We recommend parameters $[n, k]$ and t for the following security levels

- For 80-bit security against our attack we propose $[1632, 1269]$ Goppa codes (degree $t = 33$), with 34 errors added by the sender.
  Public-key size: $k \cdot (n - k) = 460647$ bits.
- Without list decoding, and restriction $n = 2^d$: $[2048, 1751]$ Goppa codes ($t = 27$). Public key size: 520047 bits.
- For 128-bit security: we propose $[2960, 2288]$ Goppa codes ($t = 56$), with 57 errors added by the sender.
  Public-key size: 1537536 bits.
- For 256-bit security: $[6624, 5129]$ Goppa codes ($t = 115$), with 117 errors added by the sender.
  Public-key size: 7667855 bits.