Two completely unrelated topics:
(1) McBits;
(2) Post-Quantum RSA

D. J. Bernstein
University of Illinois at Chicago

Thanks for (1) to:
Cisco University Research Program

Two completely unrelated topics:
(1) McBits;
(2) Post-Quantum RSA

D. J. Bernstein
University of Illinois at Chicago

Thanks for (1) to:
Cisco University Research Program

Thanks for (2) to:
No sponsors yet!

Two completely unrelated topics:
(1) McBits;
(2) Post-Quantum RSA

D. J. Bernstein
University of Illinois at Chicago

Thanks for (1) to:
Cisco University Research Program

Thanks for (2) to:
No sponsors yet! $\boxed{\text{Insert Coin}}$

Bonus topic added today:

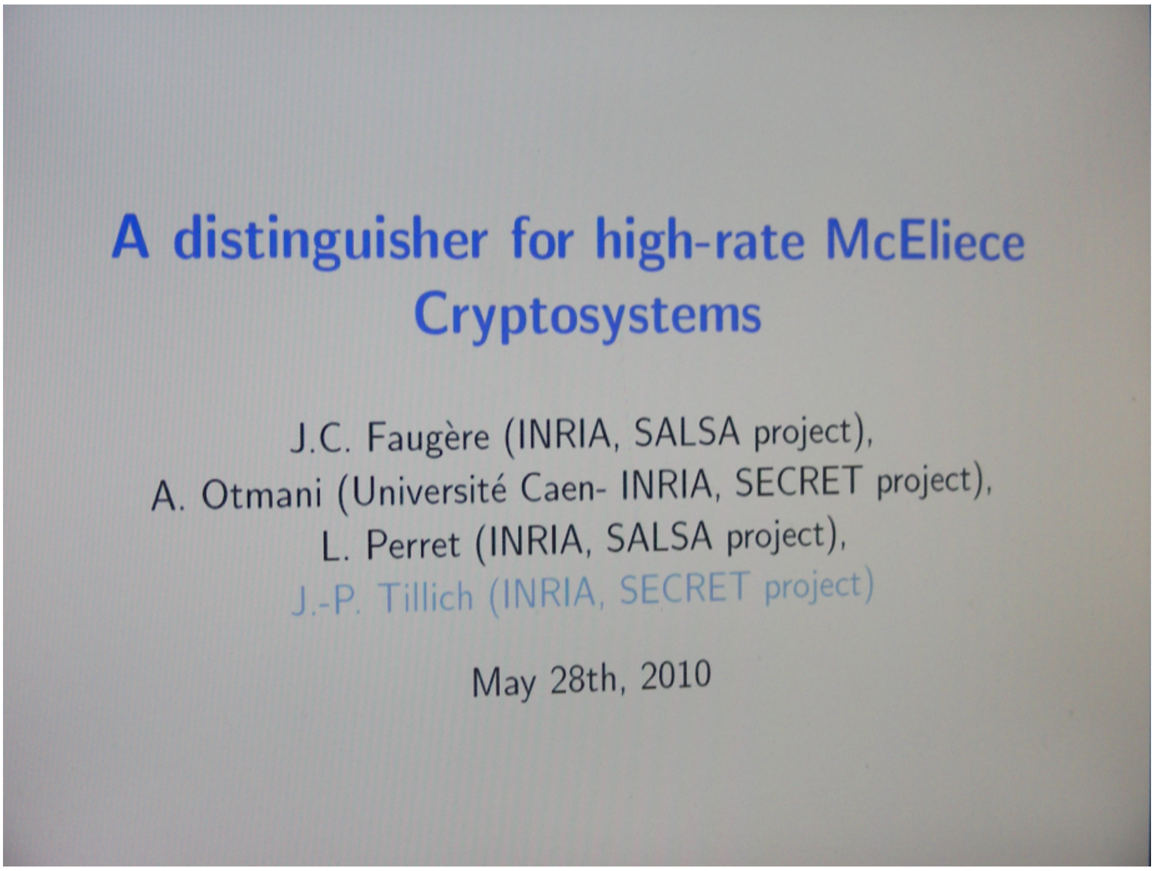
0. Wild McEliece (joint work with Tanja Lange, Christiane Peters)

Conventional wisdom on McEliece using degree-t Goppa: t errors over $\mathbf{F}_2$, but only $t/2$ errors over $\mathbf{F}_q$ if $q > 2$.

Bonus topic added today:

0. Wild McEliece (joint work with Tanja Lange, Christiane Peters)

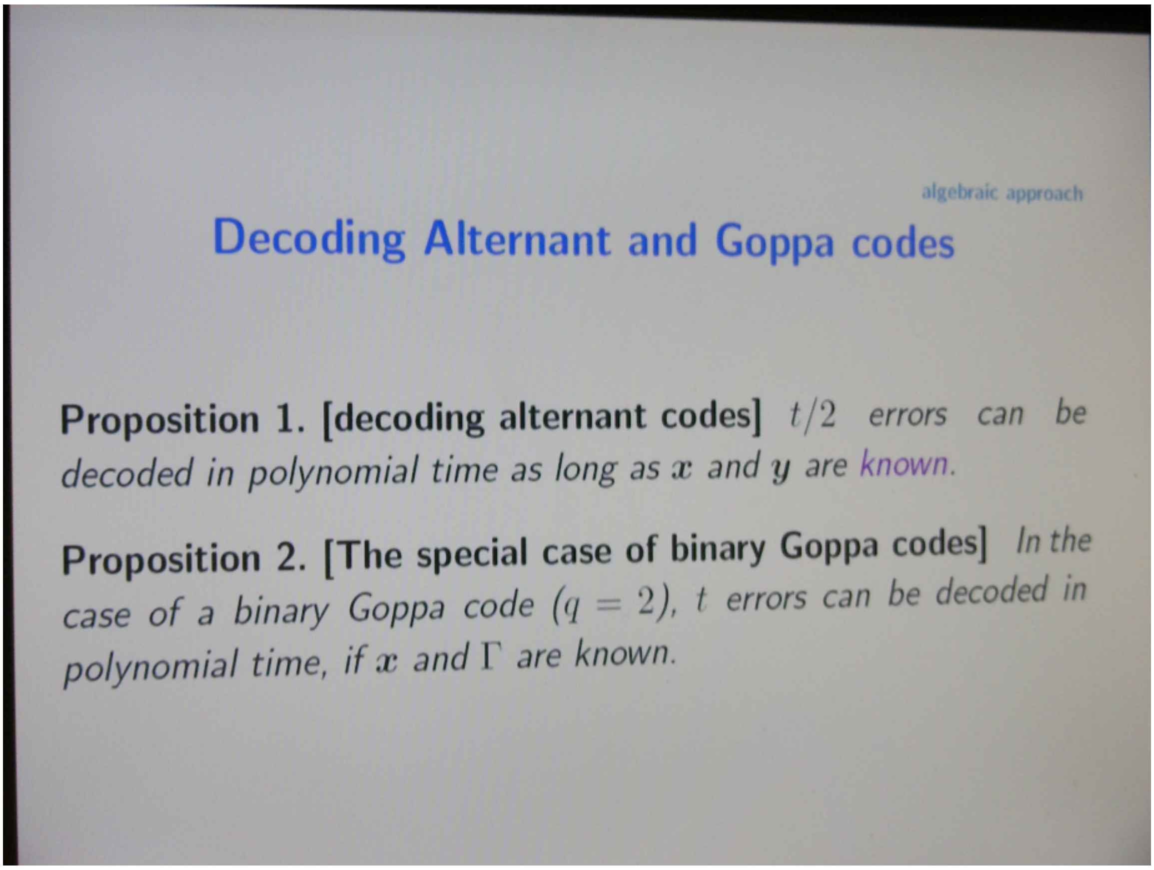
Conventional wisdom on McEliece using degree-t Goppa: t errors over $\mathbf{F}_2$, but only $t/2$ errors over $\mathbf{F}_q$ if $q > 2$.



Decoding Alternant and Goppa codes

Proposition 1. [decoding alternant codes] *$t/2$ errors can be decoded in polynomial time as long as x and y are known.*

Proposition 2. [The special case of binary Goppa codes] *In the case of a binary Goppa code ($q = 2$), t errors can be decoded in polynomial time, if x and Γ are known.*



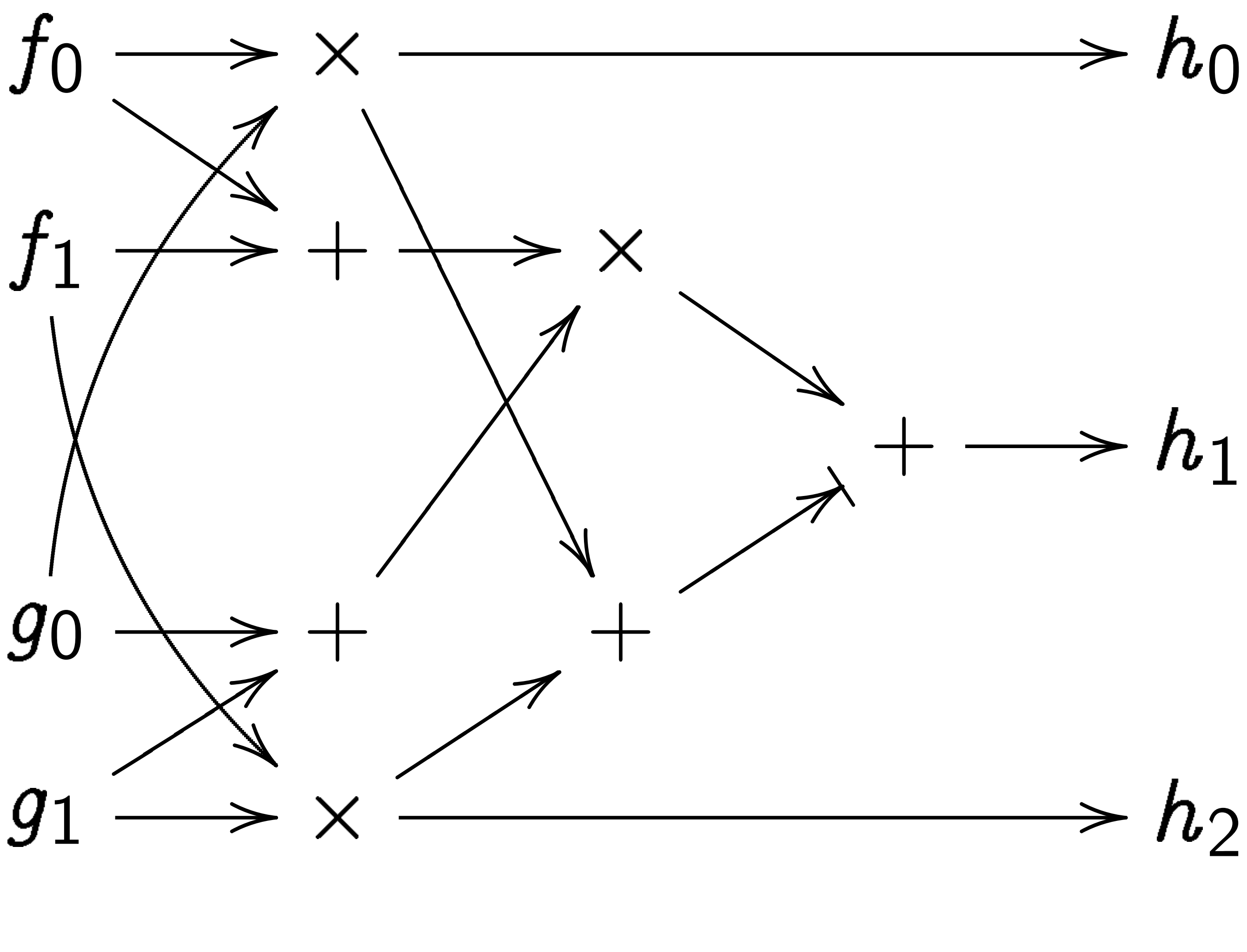
New: "Wild McEliece" uses $qt/(2(q-1))$ errors over $\mathbf{F}_q$.

More details: See talk from C. Peters from two days ago.

1. McBits: Arithmetic circuits for code-based cryptography

An $\mathbf{F}_2$-arithmetic circuit starts from inputs and constants and computes a chain of two-input $\mathbf{F}_2$-adds $u, v \mapsto u + v$, two-input $\mathbf{F}_2$-mults $u, v \mapsto uv$.

Example, not the smallest 2×2 polynomial multiplier:



What I'm working on:
fast arithmetic circuits
for confidence-inspiring
code-based public-key encryption.

Circuits are good for security:
no conditional jumps;
no variable array indices;
no input-dependent timings;
no software side channels.

Plan to publish software
and place into public domain.

Main challenge: Speed.

Metric for this project:
"ops" $=$ #adds $+$ #mults.

Clear definition; simple.

Not a bad predictor of
bitsliced software speed.

Also not a bad predictor of
throughput of unrolled hardware.

Warnings: metric doesn't see
code size ("ops" unrolls loops),
communication costs, etc.

Counting bit operations
rewards fast mult algorithms,
as in new ECC speed records
(2009 “batch binary Edwards”).
Now exploring Gao–Mateer mult.

Use fast multipoint evaluation to
eliminate conditional jumps from
fast root-finding; $n^{1+o(1)}$ ops.

Most annoying part to write:
$n^{1+o(1)}$ fast continued fraction
without conditional jumps.

Biggest asymptotic bottleneck:
matrix randomizer, $n^{2+o(1)}$ ops.
Can reduce 2 with more batching.

2. Post-Quantum RSA:
Is it possible that
the community has missed
another plausible candidate
for post-quantum cryptography?

2. Post-Quantum RSA:
Is it possible that
the community has missed
another plausible candidate
for post-quantum cryptography?

Conventional wisdom:
Shor's algorithm supersedes
all previous factorization methods.
In fact, it breaks RSA
as quickly as RSA decrypts,
so we have no hope of security
from scaling RSA key sizes.

2. Post-Quantum RSA:
Is it possible that
the community has missed
another plausible candidate
for post-quantum cryptography?

Conventional wisdom:
Shor's algorithm supersedes
all previous factorization methods.
In fact, it breaks RSA
as quickly as RSA decrypts,
so we have no hope of security
from scaling RSA key sizes.

Is this actually true?

Some methods to factor n
(assuming standard conjectures):

Trial division finds p
using $(p + \lg n)^{1+o(1)}$ bit ops.

Some methods to factor n
(assuming standard conjectures):

Trial division finds p
using $(p + \lg n)^{1+o(1)}$ bit ops.

Pollard's rho method finds p
using $(p^{1/2} \lg n)^{1+o(1)}$ bit ops.

Some methods to factor n (assuming standard conjectures):

Trial division finds p using $(p + \lg n)^{1+o(1)}$ bit ops.

Pollard's rho method finds p using $(p^{1/2} \lg n)^{1+o(1)}$ bit ops.

Quadratic sieve finds p using $(2^{(\lg n \lg \lg n)^{1/2}})^{1+o(1)}$ bit ops.

Some methods to factor n (assuming standard conjectures):

Trial division finds p using $(p + \lg n)^{1+o(1)}$ bit ops.

Pollard's rho method finds p using $(p^{1/2} \lg n)^{1+o(1)}$ bit ops.

Quadratic sieve finds p using $(2^{(\lg n \lg \lg n)^{1/2}})^{1+o(1)}$ bit ops.

ECM finds p using $(2^{(2 \lg p \lg \lg p)^{1/2}} \lg n)^{1+o(1)}$ bit ops.

Some methods to factor n (assuming standard conjectures):

Trial division finds p using $(p + \lg n)^{1+o(1)}$ bit ops.

Pollard's rho method finds p using $(p^{1/2} \lg n)^{1+o(1)}$ bit ops.

Quadratic sieve finds p using $(2^{(\lg n \lg \lg n)^{1/2}})^{1+o(1)}$ bit ops.

ECM finds p using $(2^{(2 \lg p \lg \lg p)^{1/2}} \lg n)^{1+o(1)}$ bit ops.

Number-field sieve finds p using $(2^{c(\lg n)^{1/3}(\lg \lg n)^{2/3}})^{1+o(1)}$ bit ops.

Shor's algorithm finds p
using $(\lg n)^{2+o(1)}$ qubit ops.

Let's assume that qubit ops
aren't much harder than bit ops,
and that $o(1)$ isn't very big.

Does Shor supersede NFS?
Yes.

Shor's algorithm finds p using $(\lg n)^{2+o(1)}$ qubit ops.

Let's assume that qubit ops aren't much harder than bit ops, and that $o(1)$ isn't very big.

Does Shor supersede NFS?
Yes.

Does Shor supersede ECM?
Not necessarily!

ECM beats Shor for small p: compare $2 \lg p \lg \lg p$ to $(\lg \lg n)^2$.
Best small-p algorithm I know: GEECM.

Shor's algorithm finds p using $(\lg n)^{2+o(1)}$ qubit ops.

Let's assume that qubit ops aren't much harder than bit ops, and that $o(1)$ isn't very big.

Does Shor supersede NFS?
Yes.

Does Shor supersede ECM?
Not necessarily!

ECM beats Shor for small p: compare $2 \lg p \lg \lg p$ to $(\lg \lg n)^2$.
Best small-p algorithm I know: GEECM. Grover+Edwards+ECM.

Standard RSA decryption:
compute cube root mod $n = pq$
by computing and combining
cube roots mod p and q.
$(\lg n)^{2+o(1)}$ ops.

Same as Shor. Game over?

Standard RSA decryption:
compute cube root mod $n = pq$
by computing and combining
cube roots mod p and q.
$(\lg n)^{2+o(1)}$ ops.

Same as Shor. Game over?
No! Speed up decryption.

Standard RSA decryption:
compute cube root mod $n = pq$
by computing and combining
cube roots mod p and q.
$(\lg n)^{2+o(1)}$ ops.

Same as Shor. Game over?
No! Speed up decryption.

Use “multi-prime RSA.”
1997/1998 Tandem patent

Standard RSA decryption:
compute cube root mod $n = pq$
by computing and combining
cube roots mod p and q.
$(\lg n)^{2+o(1)}$ ops.

Same as Shor. Game over?
No! Speed up decryption.

Use "multi-prime RSA."
1997/1998 Tandem patent
but already in 1983 RSA patent:
"the present invention
may use a modulus n which is
a product of three or more primes
(not necessarily distinct)."

Public key $n = p_1 p_2 \cdots p_k$.
Secret primes $p_1, p_2, \ldots, p_k$ with
$\lg p_i \in b^{2+o(1)}$, $k \in 2^{(1+o(1))b/2}$.

Key: $2^{(1+o(1))b/2}$ bits.
Encryption: $2^{(1+o(1))b/2}$ bit ops.
Decryption: $2^{(1+o(1))b/2}$ bit ops.

Shor attack, GEECM attack:
$> 2^b$ qubit ops
if each $o(1)$ was chosen properly.

Public key $n = p_1 p_2 \cdots p_k$.
Secret primes $p_1, p_2, \ldots, p_k$ with
$\lg p_i \in b^{2+o(1)}$, $k \in 2^{(1+o(1))b/2}$.

Key: $2^{(1+o(1))b/2}$ bits.
Encryption: $2^{(1+o(1))b/2}$ bit ops.
Decryption: $2^{(1+o(1))b/2}$ bit ops.

Shor attack, GEECM attack:
$> 2^b$ qubit ops
if each $o(1)$ was chosen properly.

Concrete analysis suggests that
RSA with 2^{31} 4096-bit primes
provides $> 2^{100}$ security
vs. all known quantum attacks.
Key almost fits on a hard drive.