Decoding random codes: asymptotics, benchmarks, challenges, and implementations

D. J. Bernstein
University of Illinois at Chicago

Assume that we're using
the McEliece cryptosystem
(or Neiderreiter or . . .)
to encrypt a plaintext.

Usual standard for high security:
Choose cryptosystem parameters
so that the attacker
cannot decrypt the ciphertext.

Assume that we're using
the McEliece cryptosystem
(or Neiderreiter or . . .)
to encrypt a plaintext.

Usual standard for high security:
Choose cryptosystem parameters
so that the attacker
~~cannot decrypt the ciphertext.~~
has negligible chance of
distinguishing this ciphertext
from the ciphertext
for another plaintext.

Assume that we're using
the McEliece cryptosystem
(or Neiderreiter or . . .)
to encrypt a plaintext.

Usual standard for high security:
Choose cryptosystem parameters
so that the attacker
~~cannot decrypt the ciphertext.~~
has negligible chance of
distinguishing this ciphertext
from the ciphertext
for another plaintext.

(Maybe better to account for
multi-target; see previous talk.)

Attacker's success chance increases with more computation, eventually reaching ≈ 1.

Public-key cryptography is never information-theoretically secure!

Attacker's success chance increases with more computation, eventually reaching ≈ 1.

Public-key cryptography is never information-theoretically secure!

But real-world attackers do not have unlimited computation.

The usual standard, quantified:
Choose cryptosystem parameters so that attacker has success chance at most ϵ after 2^c computations.

These parameters depend on c and ϵ.

These parameters depend on c and ϵ.

Some people assume $c < 80$.

These parameters
depend on c and ϵ.

Some people assume $c < 80$.

The ECC2K-130 project
will reach 2^{77} computations;
future projects will break 2^{80}.
Some people assume $c = 128$.

These parameters
depend on c and ϵ.

Some people assume $c < 80$.

The ECC2K-130 project
will reach 2^{77} computations;
future projects will break 2^{80}.
Some people assume $c = 128$.

Some people count #atoms
in universe. Assume $c = 384$?

These parameters
depend on c and ϵ.

Some people assume $c < 80$.

The ECC2K-130 project
will reach 2^{77} computations;
future projects will break 2^{80}.
Some people assume $c = 128$.

Some people count #atoms
in universe. Assume $c = 384$?

Less discussion of ϵ.
Is it okay for attacker
to have 1% success chance?
1/1000? 1/1000000?

How do we handle
this variability in (c, ϵ)?

Strategy 1:
A. Convince big community
to focus on one (c, ϵ),
eliminating the variability.
B. Choose parameters.

How do we handle
this variability in (c, ϵ)?

Strategy 1:
A. Convince big community
to focus on one (c, ϵ),
eliminating the variability.
B. Choose parameters.

Strategy 2—including this talk:
A. Accept the variability.
B. Choose parameters
as functions of (c, ϵ).

How do we handle
this variability in (c, ϵ)?

Strategy 1:
A. Convince big community
to focus on one (c, ϵ),
eliminating the variability.
B. Choose parameters.

Strategy 2—including this talk:
A. Accept the variability.
B. Choose parameters
as functions of (c, ϵ).

1A more complicated than 2A.

How do we handle
this variability in (c, ϵ)?

Strategy 1:
A. Convince big community
to focus on one (c, ϵ),
eliminating the variability.
B. Choose parameters.

Strategy 2—including this talk:
A. Accept the variability.
B. Choose parameters
as functions of (c, ϵ).

1A more complicated than 2A.
2B more complicated than 1B.

Helpful simplification
for code-based cryptography:

All of our best attacks
consist of many iterations.
Each iteration: small cost 2^c,
small success probability ϵ.

Separate iterations are
almost exactly independent:
$2^{c'-c}$ iterations cost $2^{c'}$,
have success probability
almost exactly $1-(1-\epsilon)^{2^{c'-c}}$.

So parameters are really just
functions of $2^c/\log(1/(1-\epsilon))$.

Is this simplification correct?

Objection 1: Is $2^{c'-c}$ an integer?

Is this simplification correct?

Objection 1: Is $2^{c'-c}$ an integer?

Response: Use $\lfloor 2^{c'-c} \rfloor$.
Iteration success probability
is so small that
we care only about $c' \gg c$.

Is this simplification correct?

Objection 1: Is $2^{c'-c}$ an integer?

Response: Use $\lfloor 2^{c'-c} \rfloor$.
Iteration success probability
is so small that
we care only about $c' \gg c$.

Objection 2: “Reusing pivots”
makes our best attacks faster
but loses some independence.

Is this simplification correct?

Objection 1: Is $2^{c'-c}$ an integer?

Response: Use $\lfloor 2^{c'-c} \rfloor$.
Iteration success probability
is so small that
we care only about $c' \gg c$.

Objection 2: "Reusing pivots"
makes our best attacks faster
but loses some independence.

Response: Yes, must replace ϵ by
result of Markov-chain analysis.

Is this simplification correct?

Objection 1: Is $2^{c'-c}$ an integer?

Response: Use $\lfloor 2^{c'-c} \rfloor$.
Iteration success probability
is so small that
we care only about $c' \gg c$.

Objection 2: “Reusing pivots”
makes our best attacks faster
but loses some independence.

Response: Yes, must replace ϵ by
result of Markov-chain analysis.
But can still merge (c, ϵ)
into $2^c / \log(1/(1-\epsilon))$.

Attacker's $2^c/\log(1/(1-\epsilon))$
depends not only on parameters
but also on attack algorithm.

Maybe attacker has found
a much faster algorithm
than anything we know!

Attacker's $2^c/\log(1/(1-\epsilon))$ depends not only on parameters but also on attack algorithm.

Maybe attacker has found a much faster algorithm than anything we know!

All public-key cryptosystems share this risk.

Attacker's $2^c/\log(1/(1-\epsilon))$
depends not only on parameters
but also on attack algorithm.

Maybe attacker has found
a much faster algorithm
than anything we know!

All public-key cryptosystems
share this risk.

Responses to this risk:
a huge amount of snake oil,
and one standard approach
that seems to be effective.

The standard approach:
Encourage many smart people
to search for speedups.

Monitor their progress:
big speedup, big speedup,
small speedup, big speedup,
small, small, tiny, big, small, tiny,
small, small, tiny, tiny, small, tiny,
tiny, small, tiny, tiny, tiny, tiny.
Eventually progress stops.

After years, build confidence
that optimal algorithm is known.

The standard approach:
Encourage many smart people
to search for speedups.

Monitor their progress:
big speedup, big speedup,
small speedup, big speedup,
small, small, tiny, big, small, tiny,
small, small, tiny, tiny, small, tiny,
tiny, small, tiny, tiny, tiny, tiny.
Eventually progress stops.

After years, build confidence
that optimal algorithm is known.
. . . or is it?

Consider cost of multiplying two n-coeff polys in $\mathbf{R}[x]$, where cost means # adds and mults in $\mathbf{R}$.

Fast Fourier transform (Gauss): $(15 + o(1))n \lg n$.

Consider cost of multiplying two
n-coeff polys in $\mathbf{R}[x]$,
where cost means
adds and mults in $\mathbf{R}$.

Fast Fourier transform (Gauss):
$(15 + o(1))n \lg n$.

Huge interest starting 1965.
Split-radix FFT (1968 Yavne):
$(12 + o(1))n \lg n$.
Many descriptions, analyses,
implementations, followups;
12 was believed optimal.

Consider cost of multiplying two n-coeff polys in $\mathbf{R}[x]$, where cost means $\#$ adds and mults in $\mathbf{R}$.

Fast Fourier transform (Gauss): $(15 + o(1))n \lg n$.

Huge interest starting 1965.
Split-radix FFT (1968 Yavne): $(12 + o(1))n \lg n$.
Many descriptions, analyses, implementations, followups; 12 was believed optimal.

Tangent FFT (2004 van Buskirk): $(34/3 + o(1))n \lg n$.

Consider cost of multiplying two n-coeff polys in $\mathbf{F}_2[x]$, where cost means $\#$ adds and mults in $\mathbf{F}_2$.

Standard schoolbook method: $2n^2 - 2n + 1$; e.g., 61 for $n = 6$.

Consider cost of multiplying two n-coeff polys in $\mathbf{F}_2[x]$, where cost means # adds and mults in $\mathbf{F}_2$.

Standard schoolbook method: $2n^2 - 2n + 1$; e.g., 61 for $n = 6$.

1963 Karatsuba method: e.g., 59 for $n = 6$.
Many descriptions, analyses, implementations, followups.
Improved for large n, but was believed optimal for small n.

Consider cost of multiplying two n-coeff polys in $\mathbf{F}_2[x]$, where cost means # adds and mults in $\mathbf{F}_2$.

Standard schoolbook method: $2n^2 - 2n + 1$; e.g., 61 for $n = 6$.

1963 Karatsuba method: e.g., 59 for $n = 6$.
Many descriptions, analyses, implementations, followups.
Improved for large n, but was believed optimal for small n.

2000 Bernstein: e.g., 57 for $n = 6$.

Consider cost of multiplying two n-bit integers in $\mathbf{Z}$, where cost means # NAND gates.

Schoolbook: $O(n^2)$.

Consider cost of multiplying two n-bit integers in $\mathbf{Z}$,
where cost means
NAND gates.

Schoolbook: $O(n^2)$.

Intense work after Karatsuba.
1971 Schönhage–Strassen:
$O(n \lg n \lg \lg n)$.
Used in many theorems.
Was believed optimal.

Consider cost of multiplying two n-bit integers in $\mathbf{Z}$, where cost means # NAND gates.

Schoolbook: $O(n^2)$.

Intense work after Karatsuba.
1971 Schönhage–Strassen: $O(n \lg n \lg\lg n)$.
Used in many theorems.
Was believed optimal.

2007 Fürer:
non-constant improvement,
almost reaching $O(n \lg n)$.

Possible conclusion 1:
We’ll never know the optimal algorithm for anything interesting.

Possible conclusion 1:
We'll never know the optimal algorithm for anything interesting.

Possible conclusion 2:
$2004 - 1968 = 36$;
$2000 - 1963 = 37$;
$2007 - 1971 = 36$.

Possible conclusion 1:
We'll never know the optimal algorithm for anything interesting.

Possible conclusion 2:
$2004 - 1968 = 36$;
$2000 - 1963 = 37$;
$2007 - 1971 = 36$.
Algorithms are optimal if they survive 38 years.

Possible conclusion 1:
We'll never know the optimal algorithm for anything interesting.

Possible conclusion 2:
$2004 - 1968 = 36$;
$2000 - 1963 = 37$;
$2007 - 1971 = 36$.
Algorithms are optimal if they survive 38 years.

Possible conclusion 3:
Should choose parameters aiming at a slightly larger c so that speedups on this scale don't compromise security.

Can also find examples
of bigger speedups
in well-studied problems, but
these examples are less common.

Reasonable to hope
that the standard approach
(encouraging many smart people
to search for speedups)
finds near-optimal attacks.

Doesn't eliminate risk,
but historical examples suggest
that the risk is much higher
for cryptosystems that do *not*
take the standard approach.

Sometimes I see papers taking steps that *discourage* this research:

1. Excessively optimistic algorithm analyses.

2. Excessively pessimistic algorithm analyses.

3. Nonsensical machine models.

Why do they do this?

Sometimes I see papers taking steps that *discourage* this research:

1. Excessively optimistic algorithm analyses.

2. Excessively pessimistic algorithm analyses.

3. Nonsensical machine models.

Why do they do this?
Napoleon: *"N'attribuez jamais à la malveillance ce qui s'explique très bien par l'incompétence."*

McEliece public key:
linear map $G : \mathbf{F}_2^k \hookrightarrow \mathbf{F}_2^n$.

McEliece plaintext:
$m \in \mathbf{F}_2^k$;
and $e \in \mathbf{F}_2^n$ of weight w.

McEliece ciphertext:
$Gm + e \in \mathbf{F}_2^n$.

Typical parameter choices:
$k = Rn$ with $R = 0.8$;
$w = (n - k)/\lceil \lg n \rceil$
$\approx (1 - R)n/\lg n$.

Basic information-set decoding,
given G and $y \in \mathbf{F}_2^n$:

Choose uniform random size-k subset $S \subseteq \{1, 2, \ldots, n\}$.

Hope that the composition $\mathbf{F}_2^k \xrightarrow{G} \mathbf{F}_2^n \to \mathbf{F}_2^S$ is invertible (S is an "information set").
If not invertible, try new S.

Project y from $\mathbf{F}_2^n$ to $\mathbf{F}_2^S$.
Apply inverse, obtaining m.
Compute $e = y - Gm$.
If weight of e is not w, try new S.

Idea introduced by 1962 Prange.

Easy to analyze speed of one iteration (one choice of S): Gaussian elimination to invert $k \times k$ matrix; matrix-vector multiplication; etc.

Easy to analyze probability for almost all choices of G: $0.288\ldots$ chance of invertibility; $\binom{n-k}{w}/\binom{n}{w}$ chance that e is 0 on $\mathbf{F}_2^S$; overall iteration success chance $0.288\ldots\binom{n-k}{w}/\binom{n}{w}$.

1978 McEliece repeats same idea but has different analysis:

"A more promising attack is to select k of the n coordinates randomly in hope that none of the k are in error . . . The probability of no error, however, is about $(1 - \frac{t}{n})^k$, and the amount of work involved in solving . . . is about k^3. . . . one expects a work factor of $k^3 \cdot (1 - \frac{t}{n})^{-k}$. For $n = 1024$, $k = 524$, $t = 50$ this is about $10^{19} \approx 2^{65}$."

McEliece probability analysis was excessively optimistic; lazy approximations are too small.

1988 Adams–Meijer:
$$k^3 \Big/ \left(0.288\ldots \binom{n-k}{w} / \binom{n}{w}\right) \approx 2^{83}.$$

McEliece probability analysis
was excessively optimistic;
lazy approximations are too small.

1988 Adams–Meijer:
$$k^3 \Big/ \left(0.288\ldots \binom{n-k}{w} / \binom{n}{w}\right) \approx 2^{83}.$$

How can someone publish
an interesting new speedup
from 2^{83} to 2^{73},
if McEliece said 2^{65}?

Extra work for authors
to convince reviewers
that McEliece was wrong.
Where's McEliece's erratum?

1999 Barg et al.: Huge speedups from "supercode decoding."

Cost $2^{(0.101\ldots+o(1))n}$ if $n \to \infty$, assuming $w/n \to W$ and $k/n \to 1/2 = 1 + W \lg W + (1-W)\lg(1-W)$.

Best previous result: Cost $2^{(0.115\ldots+o(1))n}$.

1999 Barg et al.: Huge speedups from "supercode decoding."

Cost $2^{(0.101\ldots+o(1))n}$ if $n \to \infty$, assuming $w/n \to W$ and $k/n \to 1/2 = 1 + W \lg W + (1-W)\lg(1-W)$.

Best previous result: Cost $2^{(0.115\ldots+o(1))n}$.

But 1999 Barg et al. is wrong! Critical error in "Corollary 12" kills analysis and conclusions. Mentioned in Crypto 2011 paper by Bernstein–Lange–Peters.

2009 Finiasz–Sendrier:

"To evaluate the cost of the algorithm we will assume that only the instructions (ISD i) are significant. . . . It is a valid assumption as we only want a lower bound. . . . $\mathrm{WF}_{\mathrm{ISD}} \approx \cdots$"

2009 Finiasz–Sendrier:

"To evaluate the cost of the algorithm we will assume that only the instructions (ISD i) are significant. . . . It is a valid assumption as we only want a lower bound. . . . $\mathrm{WF}_{\mathrm{ISD}} \approx \cdots$"

No, a lower bound is not enough! Need to state actual attack cost to encourage future research. Lower bound is too optimistic, discourages future research.

Research is also discouraged by excessively *pessimistic* algorithm analyses.

Real speedups are unrecognized, unadvertised, abandoned.

Research is also discouraged by excessively *pessimistic* algorithm analyses.
Real speedups are unrecognized, unadvertised, abandoned.

2011 Bernstein–Lange–Peters found most important idea in "ball-collision decoding" by analyzing supercode decoding.

2009 Finiasz–Sendrier missed speedup because they had an overly pessimistic analysis: lazy approximation $\binom{k+\ell}{p} \approx \binom{k}{p}$.

Perhaps the biggest drain
on research in this area:
nonsensical machine models.

Perhaps the biggest drain on research in this area: nonsensical machine models.

1998 Canteaut–Chabaud: "We give here an explicit and computable expression for the work factor of this algorithm, i.e., the average number of elementary operations it requires."

How can someone write a followup paper demonstrating a smaller "work factor"? Where is the definition of "elementary operations"?

Canteaut et al. obviously
aren't counting memory access,
copies, communication costs.
"Elementary operations"
are fully explained by arithmetic.

Write speedup paper that
counts these "operations"
and doesn't count memory access.

Canteaut et al. obviously
aren't counting memory access,
copies, communication costs.
"Elementary operations"
are fully explained by arithmetic.

Write speedup paper that
counts these "operations"
and doesn't count memory access.

Reviewer: "When the authors
compute the complexity of one
iteration of the algorithm they
neglect (or deliberately forget) the
cost of the join operation between
the sets S and T."

How do we get out of this mess?
Surely we can cite definitions
from computational complexity?

Typical “RAM” definition?

How do we get out of this mess?
Surely we can cite definitions
from computational complexity?

Typical “RAM” definition?
Nonsensical results: can do
$\Theta(n^2)$ bit ops in “time” n.
Okay for poly-time theorems,
not for serious optimization.

How do we get out of this mess?
Surely we can cite definitions
from computational complexity?

Typical "RAM" definition?
Nonsensical results: can do
$\Theta(n^2)$ bit ops in "time" n.
Okay for poly-time theorems,
not for serious optimization.

"Pointer machines"—
much more restrictive?

How do we get out of this mess?
Surely we can cite definitions
from computational complexity?

Typical “RAM” definition?
Nonsensical results: can do
$\Theta(n^2)$ bit ops in “time” n.
Okay for poly-time theorems,
not for serious optimization.

“Pointer machines”—
much more restrictive?
1980 Schönhage:
Can multiply n-bit integers
in $\Theta(n)$ operations
on a pointer machine.

Count # NANDs in a circuit?

Mathematically pleasing.

Not obviously nonsensical.

Count # NANDs in a circuit?

Mathematically pleasing.
Not obviously nonsensical.

Circuits have fixed connections.
Simulate RAM by sorting.
Some work, but reasonably easy.

Count # NANDs in a circuit?

Mathematically pleasing.
Not obviously nonsensical.

Circuits have fixed connections.
Simulate RAM by sorting.
Some work, but reasonably easy.

Still physically unrealizable:
ignores wire delay, wire cost.

Count # NANDs in a circuit?

Mathematically pleasing.
Not obviously nonsensical.

Circuits have fixed connections.
Simulate RAM by sorting.
Some work, but reasonably easy.

Still physically unrealizable:
ignores wire delay, wire cost.

1981 Brent–Kung AT theorem:
n-bit multiplication on
realistic size-n parallel circuit
has to take time $n^{1/2}$
even without wire delay.

A few suggestions

Want correct analyses
in clear cost metrics.

Brent–Kung: realistic;
not excessively complicated;
suitable for asymptotics.

A few suggestions

Want correct analyses
in clear cost metrics.

Brent–Kung: realistic;
not excessively complicated;
suitable for asymptotics.

NANDs: trades realism
for attractive simplicity;
suitable for asymptotics.

<u>A few suggestions</u>

Want correct analyses
in clear cost metrics.

Brent–Kung: realistic;
not excessively complicated;
suitable for asymptotics.

NANDs: trades realism
for attractive simplicity;
suitable for asymptotics.

Time on CPU X: realistic;
not as easy; not asymptotic;
allows computer verification.

RSA factoring challenges
have encouraged and recognized
progress in integer factorization.

Several new attempts to do this
for post-quantum cryptography.

Mistakes to learn from:
ECC challenges
are too widely spaced;
ECC and RSA solutions
don't measure time.

2011 Bernstein–Lange–Peters:
new "partly wild" challenges,
reasonably tight spacing;
will keep track of time.

```
q = 13

m = 3

n = 451

s = 24

t = 2

u = 48

k = 307

w = 25

ciphertext = [11, 9, 12, 11, 10, ..., 1, 11

recovered_plaintext_using_secret_key = True

pubkeycol144 = [0, 10, 3, 5, 1, ..., 4, 12,

pubkeycol145 = [12, 6, 8, 3, 2, ..., 10, 7,

...

pubkeycol450 = [7, 10, 8, 10, 11, ..., 11,
```