Cryptanalysis using GPUs

Daniel J. Bernstein[2] Tanja Lange[1]

[1]Technische Universiteit Eindhoven

[2]University of Illinois at Chicago

16 May 2018

Security in Times of Surveillance

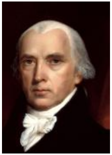
https://www.win.tue.nl/eipsi/surveillance.html

Cryptography

- Motivation #1: Communication channels are spying on our data.
- Motivation #2: Communication channels are modifying our data.

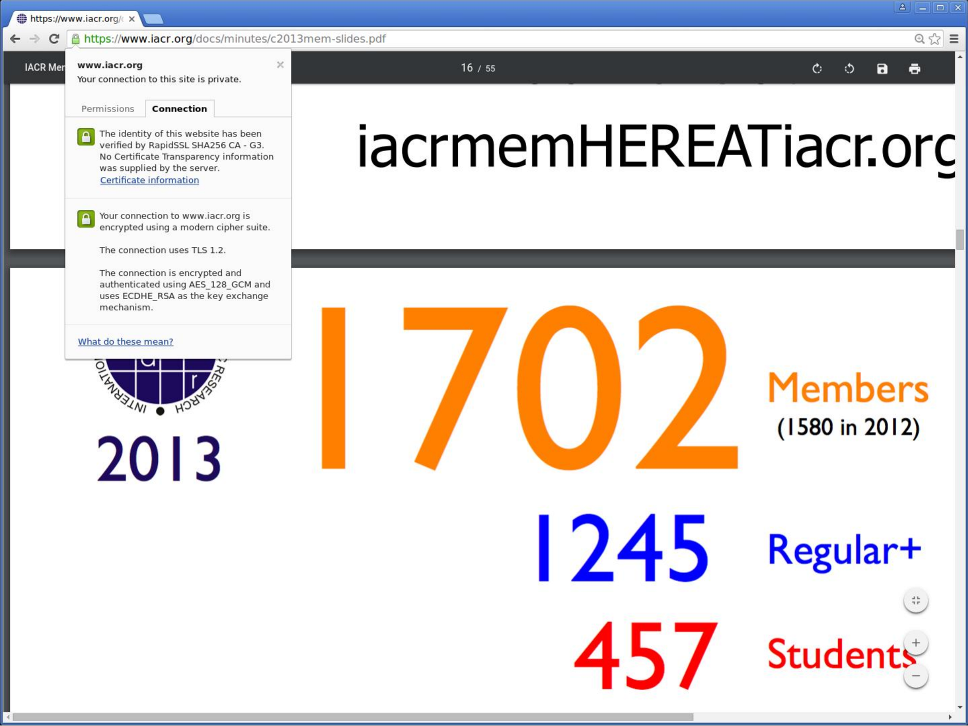
Sender
"Jefferson"

Untrustworthy network
"Eavesdropper"



Receiver
"Madison"

- Literal meaning of cryptography: "secret writing".
- Achieves various security goals by secretly transforming messages.


https://www.iacr.org/docs/minutes/c2013mem-slides.pdf
IACR Mer
16 / 55
www.iacr.org
Your connection to this site is private.
Permissions
Connection
The identity of this website has been verified by RapidSSL SHA256 CA - G3. No Certificate Transparency information was supplied by the server.
Certificate information
Your connection to www.iacr.org is encrypted using a modern cipher suite.
The connection uses TLS 1.2.
The connection is encrypted and authenticated using AES_128_GCM and uses ECDHE_RSA as the key exchange mechanism.
What do these mean?
iacrmemHEREATiacr.org
2013
1702
Members
(1580 in 2012)
1245
Regular+
457
Students


https://www.iacr.org/
https://www.iacr.org/docs/minutes/c2013mem-slides.pdf
IACR Mer
16 / 55
www.iacr.org
Your connection to this site is private.
Permissions
Connection
The identity of this website has been verified by RapidSSL SHA256 CA - G3. No Certificate Transparency information was supplied by the server.
Certificate information
Your connection to www.iacr.org is encrypted using a modern cipher suite.
The connection uses TLS 1.2.
The connection is encrypted and authenticated using AES_128_GCM and uses ECDHE_RSA as the key exchange mechanism.
What do these mean?
iacrmemH
170

Secret-key encryption

- Prerequisite: Jefferson and Madison share a secret key .
- Prerequisite: Eve doesn't know .
- Jefferson and Madison exchange any number of messages.
- Security goal #1: **Confidentiality** despite Eve's espionage.

Secret-key authenticated encryption

- Prerequisite: Jefferson and Madison share a secret key .
- Prerequisite: Eve doesn't know .
- Jefferson and Madison exchange any number of messages.
- Security goal #1: **Confidentiality** despite Eve's espionage.
- Security goal #2: **Integrity**, i.e., recognizing Eve's sabotage.

Secret-key authenticated encryption

- Prerequisite: Jefferson and Madison share a secret key .
- Prerequisite: Eve doesn't know .
- Jefferson and Madison exchange any number of messages.
- Security goal #1: **Confidentiality** despite Eve's espionage.
- Security goal #2: **Integrity**, i.e., recognizing Eve's sabotage.

Security considerations

- A and B use a shared key k in an encryption algorithm.
- Keys are typically strings of bits $k \in \{0, 1\}$.
- How long does k have to be?

Security considerations

- A and B use a shared key k in an encryption algorithm.
- Keys are typically strings of bits $k \in \{0, 1\}$.
- How long does k have to be?
- Good symmetric ciphers require the attacker to do 2^n operations.

Security considerations

- A and B use a shared key k in an encryption algorithm.
- Keys are typically strings of bits $k \in \{0, 1\}$.
- How long does k have to be?
- Good symmetric ciphers require the attacker to do 2^n operations.
- What is an operation here? How long does an operation take?

Security considerations

- A and B use a shared key k in an encryption algorithm.
- Keys are typically strings of bits $k \in \{0,1\}$.
- How long does k have to be?
- Good symmetric ciphers require the attacker to do 2^n operations.
- What is an operation here? How long does an operation take?
- Typically an operation is an execution of the encryption algorithm; this means brute force search through the entire keyspace.

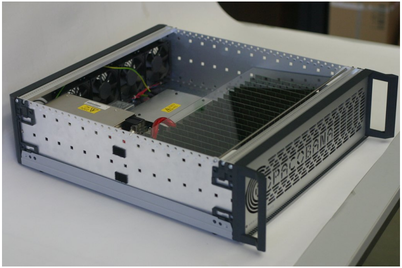
Cost of attacks

- The current standard symmetric encryption is AES (Advanced Encryption Standard).
- AES exists in three versions: AES-128, AES-192, AES-256, where AES-n means the key has n bits.
- Older standards are DES (Data Encryption Standard) and 3-DES.
- DES has $n = 56$, each DES run is pretty cheap – is this cheap enough to just break?

Cost of attacks

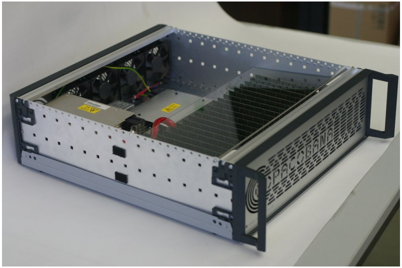
- The current standard symmetric encryption is AES (Advanced Encryption Standard).
- AES exists in three versions: AES-128, AES-192, AES-256, where AES-n means the key has n bits.
- Older standards are DES (Data Encryption Standard) and 3-DES.
- DES has $n = 56$, each DES run is pretty cheap – is this cheap enough to just break?

- SHARCS 2006
  "How to Break DES for EUR 8,980"
  built FPGA cluster COPACOBANA.
- Today: easily done on GPU cluster, paid service available online.
- So, what should n be?



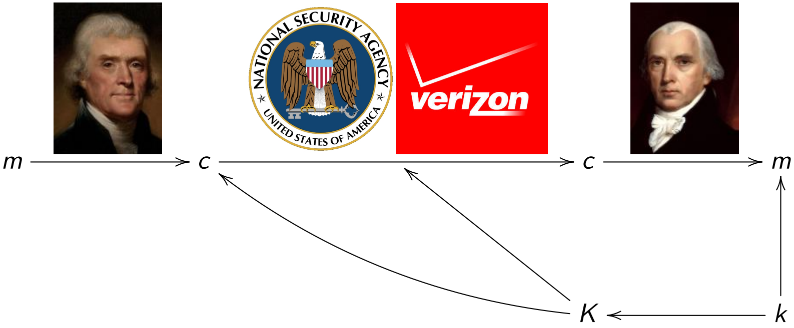
Cost of attacks

- The current standard symmetric encryption is AES (Advanced Encryption Standard).
- AES exists in three versions: AES-128, AES-192, AES-256, where AES-n means the key has n bits.
- Older standards are DES (Data Encryption Standard) and 3-DES.
- DES has $n = 56$, each DES run is pretty cheap – is this cheap enough to just break?



- SHARCS 2006
  "How to Break DES for EUR 8,980"
  built FPGA cluster COPACOBANA.
- Today: easily done on GPU cluster, paid service available online.
- So, what should n be?
- Sure larger than 56!
  For everything else:
  Depends on speed of encryption if we want to cut it close
  (or just use AES-256).

Public-key encryption



- Alice uses Bob's public key K to encrypt.
- Bob uses his secret key k to decrypt.
- Computational assumption is that recovering k from K is hard.
- Systems are a lot more complex, typically faster to break than with brute force.

Discrete logarithms on elliptic curves

- Systems work in a group, so there is some operation $+$.
- Denote $\underbrace{P + P + \cdots + P}_{a \text{ copies}} = aP$. Work in $\langle P \rangle = \{aP | a \in \mathbf{Z}\}$.
- Discrete Logarithm Problem: Given P and $Q = aP$, find a.
- Discrete logarithms are one of the main categories in public-key cryptography.
- Elliptic curves over finite fields provide good groups for cryptography.
- Group with $\approx 2^n$ elements needs $\approx 2^{n/2}$ operations to break.
- One operation typically more expensive than DES or AES.
- Lots of optimization targets for the attack:
  - Computations in the finite field.
  - Computations on the elliptic curve.
  - The main attack.

Pollard's rho method

- Make a pseudo-random walk in $\langle P \rangle$, where the next step depends on current point: $P_{i+1} = f(P_i)$.
- Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.
- The walk has now entered a cycle.
  Cycle-finding algorithm (e.g., Floyd) quickly detects this.

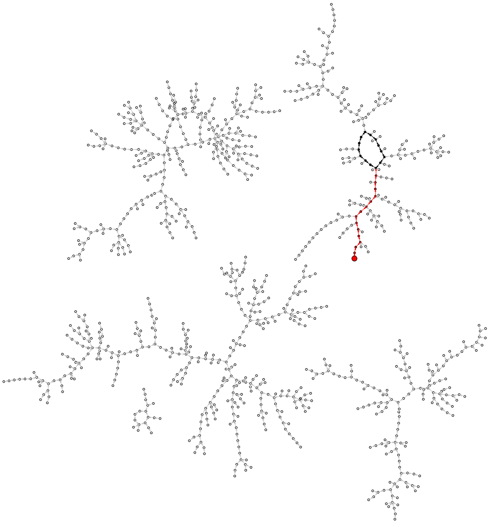
Pollard's rho method

- Make a pseudo-random walk in $\langle P \rangle$, where the next step depends on current point: $P_{i+1} = f(P_i)$.
- Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.
- The walk has now entered a cycle.
  Cycle-finding algorithm (e.g., Floyd) quickly detects this.
- Assume that for each point we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$ so that $P_i = [a_i]P + [b_i]Q$. Then $P_i = P_j$ means that

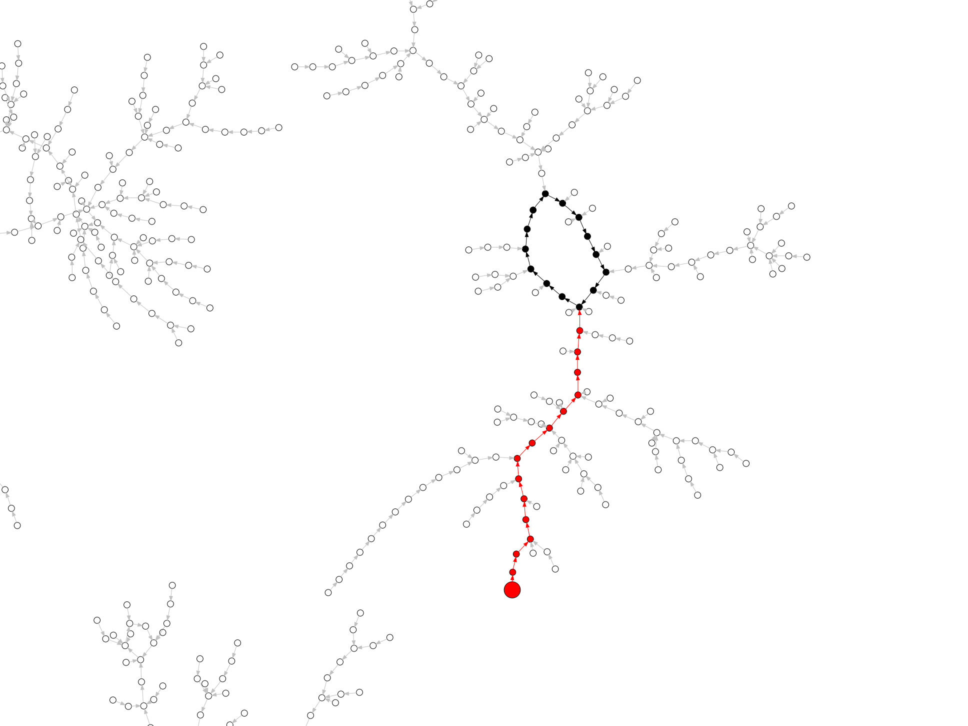
$$[a_i]P + [b_i]Q = [a_j]P + [b_j]Q \quad \text{so} \quad [b_i - b_j]Q = [a_j - a_i]P.$$

- If $b_i \neq b_j$ the ECDLP is solved: $k = (a_j - a_i)/(b_i - b_j)$ modulo ℓ.

A rho within a random walk on 1024 elements



Method is called rho method because of the shape.



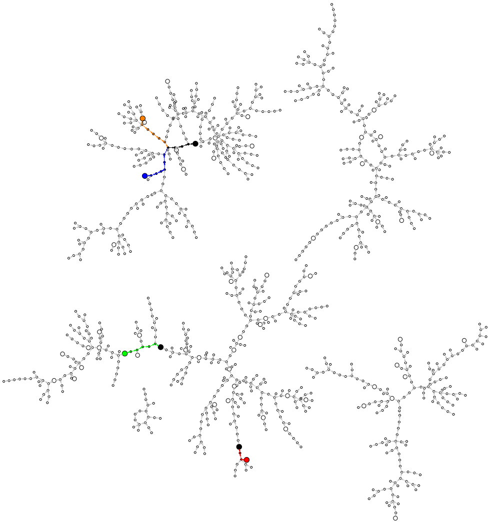
Parallel collision search

- Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.
- Want better way to spread computation across clients.
  Want to find collisions between walks on different machines, without frequent synchronization!

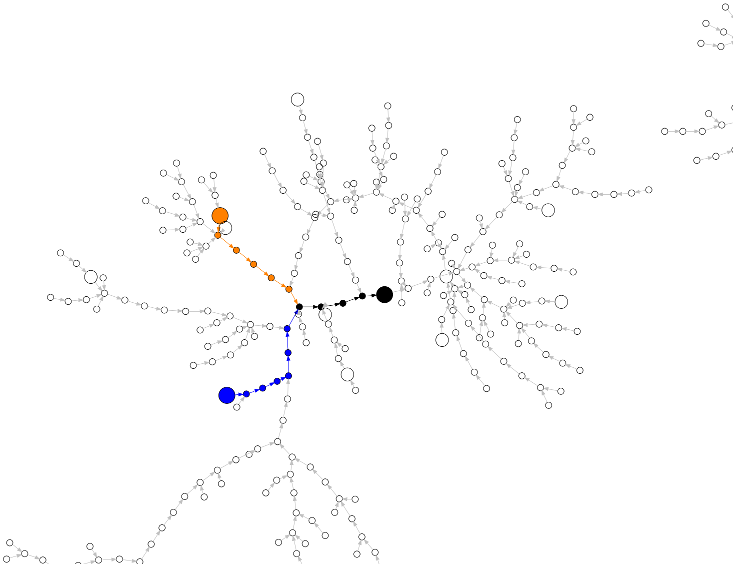
Parallel collision search

- Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.
- Want better way to spread computation across clients. Want to find collisions between walks on different machines, without frequent synchronization!
- Perform walks with different starting points but same update function on all computers. If same point is found on two different computers also the following steps will be the same.
- Terminate each walk once it hits a distinguished point. Attacker chooses definition of distinguished points; can be more or less frequent. Do not wait for cycle.
- Collect all distinguished points in central database.
- Expect collision within $O(\sqrt{\ell}/N)$ iterations. Speedup $\approx N$.

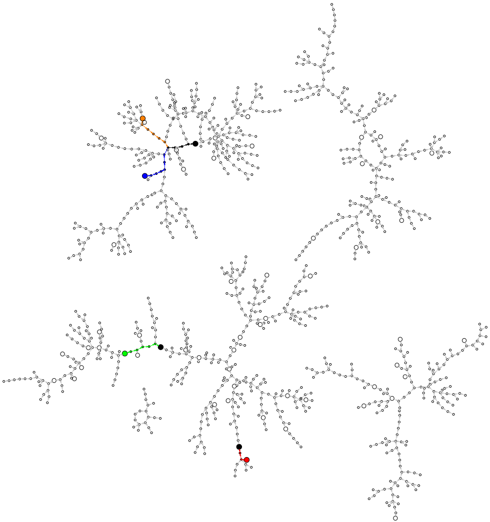
Short walks ending in distinguished points



Blue and orange paths found the same distinguished point!



Short walks ending in distinguished points



Blue and orange paths found the same distinguished point!

Some tastes of problems

- "Adding walk": Start with $P_0 = P$ and put $f(P_i) = P_i + [c_r]P + [d_r]Q$ where $r = h(P_i)$ and image of h is small. Precompute $[c_i]P + [d_i]Q$, take only one addition per step.
- P and $-P$ can be identified. Search for collisions on these classes. Search space for collisions is only $\ell/2$; this gives factor $\sqrt{2}$ speedup ... provided that $f(P_i) = f(-P_i)$.
- Solution: $f(P_i) = |P_i| + [c_r]P + [d_r]Q$ where $r = h(|P_i|)$. Define $|P_i|$ as, e.g., lexicographic minimum of $P_i, -P_i$.

Some tastes of problems

- "Adding walk": Start with $P_0 = P$ and put $f(P_i) = P_i + [c_r]P + [d_r]Q$ where $r = h(P_i)$ and image of h is small. Precompute $[c_i]P + [d_i]Q$, take only one addition per step.
- P and $-P$ can be identified. Search for collisions on these classes. Search space for collisions is only $\ell/2$; this gives factor $\sqrt{2}$ speedup ... provided that $f(P_i) = f(-P_i)$.
- Solution: $f(P_i) = |P_i| + [c_r]P + [d_r]Q$ where $r = h(|P_i|)$. Define $|P_i|$ as, e.g., lexicographic minimum of $P_i, -P_i$.
- Problem: this walk can run into fruitless cycles! If there are S different steps $[c_r]P + [d_r]Q$ then with probability $1/(2S)$ the following happens for some step:

$$\begin{aligned} P_{i+2} &= P_{i+1} + [c_r]P + [d_r]Q \\ &= -(P_i + [c_r]P + [d_r]Q) + [c_r]P + [d_r]Q = -P_i, \end{aligned}$$

  i.e. $|P_i| = |P_{i+2}|$. Get $|P_{i+3}| = |P_{i+1}|$, $|P_{i+4}| = |P_i|$, etc.
- Can detect and fix, but requires attention.
- Probability of success was computed incorrectly for years; scaling depends on many factors.

Interlude: Optimizing compilers vs. humans

"By the late 1990s for even performance sensitive code, optimizing compilers exceeded the performance of human experts."

Interlude: Optimizing compilers vs. humans

"By the late 1990s for even performance sensitive code, optimizing compilers exceeded the performance of human experts."

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

Interlude: Optimizing compilers vs. humans

"By the late 1990s for even performance sensitive code, optimizing compilers exceeded the performance of human experts."

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

— The experts disagree, and hold the speed records.

Interlude: Optimizing compilers vs. humans

"By the late 1990s for even performance sensitive code, optimizing compilers exceeded the performance of human experts."

"We come so close to optimal on most architectures that we can't do much more without using NP complete algorithms instead of heuristics. We can only try to get little niggles here and there where the heuristics get slightly wrong answers."

— The experts disagree, and hold the speed records.

"Which compiler is this which can, for instance, take Netlib LAPACK and run serial Linpack as fast as OpenBLAS on recent x86-64? (Other common hotspots are available.) Enquiring HPC minds want to know."

Why are compilers not catching up?

The actual machine is evolving farther and farther away from the source machine used by, e.g., C programs:

- Pipelining.
- Superscalar processing.
- Vectorization.
- Many threads; many cores.
- The memory hierarchy; the ring; the mesh.
- Larger-scale parallelism.
- Larger-scale networking.

Why are compilers not catching up?

The actual machine is evolving farther and farther away
from the source machine used by, e.g., C programs:

- Pipelining.
- Superscalar processing.
- Vectorization.
- Many threads; many cores.
- The memory hierarchy; the ring; the mesh.
- Larger-scale parallelism.
- Larger-scale networking.

Can reduce compiler difficulties by changing the source machine.

Why are compilers not catching up?

The actual machine is evolving farther and farther away
from the source machine used by, e.g., C programs:

- Pipelining.
- Superscalar processing.
- Vectorization.
- Many threads; many cores.
- The memory hierarchy; the ring; the mesh.
- Larger-scale parallelism.
- Larger-scale networking.

Can reduce compiler difficulties by changing the source machine.
CUDA lets programmer explicitly state parallelization, vectorization.

Why are compilers not catching up?

The actual machine is evolving farther and farther away
from the source machine used by, e.g., C programs:

- Pipelining.
- Superscalar processing.
- Vectorization.
- Many threads; many cores.
- The memory hierarchy; the ring; the mesh.
- Larger-scale parallelism.
- Larger-scale networking.

Can reduce compiler difficulties by changing the source machine.
CUDA lets programmer explicitly state parallelization, vectorization.
But still problems with instruction scheduling, register allocation.

ECC2K-130 on NVIDIA GTX 295 graphics cards

70110 bit operations in one ECC2K-130 iteration: XOR, XOR, AND, . . .

Target: NVIDIA GTX 295 dual-GPU graphics card.
60 MPs, each with 8 32-bit ALUs running at 1.242GHz.
Each ALU takes one cycle for (e.g.) 32-bit XOR.

ECC2K-130 on NVIDIA GTX 295 graphics cards

70110 bit operations in one ECC2K-130 iteration: XOR, XOR, AND, . . .

Target: NVIDIA GTX 295 dual-GPU graphics card.
60 MPs, each with 8 32-bit ALUs running at 1.242GHz.
Each ALU takes one cycle for (e.g.) 32-bit XOR.

Lower bound for 70110 bit operations on one MP:

ECC2K-130 on NVIDIA GTX 295 graphics cards

70110 bit operations in one ECC2K-130 iteration: XOR, XOR, AND, ...

Target: NVIDIA GTX 295 dual-GPU graphics card.
60 MPs, each with 8 32-bit ALUs running at 1.242GHz.
Each ALU takes one cycle for (e.g.) 32-bit XOR.

Lower bound for 70110 bit operations on one MP: 273 cycles.

ECC2K-130 on NVIDIA GTX 295 graphics cards

70110 bit operations in one ECC2K-130 iteration: XOR, XOR, AND, . . .

Target: NVIDIA GTX 295 dual-GPU graphics card.
60 MPs, each with 8 32-bit ALUs running at 1.242GHz.
Each ALU takes one cycle for (e.g.) 32-bit XOR.

Lower bound for 70110 bit operations on one MP: 273 cycles.

Best speed we obtained with NVIDIA compilers:

ECC2K-130 on NVIDIA GTX 295 graphics cards

70110 bit operations in one ECC2K-130 iteration: XOR, XOR, AND, . . .

Target: NVIDIA GTX 295 dual-GPU graphics card.
60 MPs, each with 8 32-bit ALUs running at 1.242GHz.
Each ALU takes one cycle for (e.g.) 32-bit XOR.

Lower bound for 70110 bit operations on one MP: 273 cycles.

Best speed we obtained with NVIDIA compilers: $\approx$3000 cycles.
Constantly running out of registers; huge cost for spills.

ECC2K-130 on NVIDIA GTX 295 graphics cards

70110 bit operations in one ECC2K-130 iteration: XOR, XOR, AND, ...

Target: NVIDIA GTX 295 dual-GPU graphics card.
60 MPs, each with 8 32-bit ALUs running at 1.242GHz.
Each ALU takes one cycle for (e.g.) 32-bit XOR.

Lower bound for 70110 bit operations on one MP: 273 cycles.

Best speed we obtained with NVIDIA compilers: $\approx$3000 cycles.
Constantly running out of registers; huge cost for spills.

Best speed we obtained with NVIDIA's `ptxas` assembler:

ECC2K-130 on NVIDIA GTX 295 graphics cards

70110 bit operations in one ECC2K-130 iteration: XOR, XOR, AND, ...

Target: NVIDIA GTX 295 dual-GPU graphics card.
60 MPs, each with 8 32-bit ALUs running at 1.242GHz.
Each ALU takes one cycle for (e.g.) 32-bit XOR.

Lower bound for 70110 bit operations on one MP: 273 cycles.

Best speed we obtained with NVIDIA compilers: ≈3000 cycles.
Constantly running out of registers; huge cost for spills.

Best speed we obtained with NVIDIA's `ptxas` assembler: ≈3000 cycles.
PTX is *not* the machine language. Converts to SSA, re-assigns regs.

ECC2K-130 on NVIDIA GTX 295 graphics cards

70110 bit operations in one ECC2K-130 iteration: XOR, XOR, AND, . . .

Target: NVIDIA GTX 295 dual-GPU graphics card.
60 MPs, each with 8 32-bit ALUs running at 1.242GHz.
Each ALU takes one cycle for (e.g.) 32-bit XOR.

Lower bound for 70110 bit operations on one MP: 273 cycles.

Best speed we obtained with NVIDIA compilers: $\approx$3000 cycles.
Constantly running out of registers; huge cost for spills.

Best speed we obtained with NVIDIA's `ptxas` assembler: $\approx$3000 cycles.
PTX is *not* the machine language. Converts to SSA, re-assigns regs.

Starting from van der Laan's `decuda` and `cudasm`,
we built a new assembly language:

ECC2K-130 on NVIDIA GTX 295 graphics cards

70110 bit operations in one ECC2K-130 iteration: XOR, XOR, AND, . . .

Target: NVIDIA GTX 295 dual-GPU graphics card.
60 MPs, each with 8 32-bit ALUs running at 1.242GHz.
Each ALU takes one cycle for (e.g.) 32-bit XOR.

Lower bound for 70110 bit operations on one MP: 273 cycles.

Best speed we obtained with NVIDIA compilers: ≈3000 cycles.
Constantly running out of registers; huge cost for spills.

Best speed we obtained with NVIDIA's `ptxas` assembler: ≈3000 cycles.
PTX is *not* the machine language. Converts to SSA, re-assigns regs.

Starting from van der Laan's `decuda` and `cudasm`,
we built a new assembly language: 1164 cycles.
Still many loads and stores, but much better than before.

What does the assembly language look like?

C/C++/CUDA:

```
z2 = x2 ^ y2;
```

What does the assembly language look like?

C/C++/CUDA:

```
z2 = x2 ^ y2;
```

PTX, not a true assembly language:

```
xor.b32 %r24, %r22, %r23;
```

What does the assembly language look like?

C/C++/CUDA:

```
z2 = x2 ^ y2;
```

PTX, not a true assembly language:

```
xor.b32 %r24, %r22, %r23;
```

`cudasm`:

```
xor.b32 $r2, $r3, $r2
```

What does the assembly language look like?

C/C++/CUDA:

```
z2 = x2 ^ y2;
```

PTX, not a true assembly language:

```
xor.b32 %r24, %r22, %r23;
```

`cudasm`:

```
xor.b32 $r2, $r3, $r2
```

Our `qhasm-cudasm`:

```
z2 = x2 ^ y2
```

What does the assembly language look like?

C/C++/CUDA:

```
z2 = x2 ^ y2;
```

PTX, not a true assembly language:

```
xor.b32 %r24, %r22, %r23;
```

`cudasm`:

```
xor.b32 $r2, $r3, $r2
```

Our `qhasm-cudasm`:

```
z2 = x2 ^ y2
```

For more information:
Bernstein–Chen–Cheng–Lange–Niederhagen–Schwabe–Yang
"Usable assembly language for GPUs: a success story".

Other GPU projects

- Integer factorization, in particular ECM.
- Computations of hash functions:
  - Approximate preimages (most positions match in the output).
  - Disproving DNSSEC confidentiality claims.
  - Study of backdoorability of elliptic curves.
- Cryptanalysis of post-quantum cryptography,
  see Kai-Chun Ning's talk for an example.
- Saber cluster:
  24 PCs with AMD FX-8350,
  each 32GB RAM and 2 GTX-780.
  Assembled in our very own
  sweatshop.

Security in Times of Surveillance

https://www.win.tue.nl/eipsi/surveillance.html