Bad directions in cryptographic hash functions

Daniel J. Bernstein, Andreas Hülsing, Tanja Lange,
and Ruben Niederhagen

July 1, 2015

Challenge announced at CRYPTO 2014 rump session:

Page 11 from `http://crypto.2014.rump.cr.yp.to/bca480a4e7fcdaf5bfa9dec75ff890c8.pdf`:

Everybody loves (virtual black-box / indistinguishability) obfuscation... so we implemented it!

Implementation combines ideas from various obfuscation papers and uses CLT multilinear map scheme

It is slow... but not as slow as you might think

Example: To obfuscate a 16-bit point function (i.e., 16 OR gates) with 52 bits of security using an Amazon EC2 machine with 32 cores:

- Obfuscation time: $\approx$ 7 hours
- Evaluation time: $\approx$ 3 hours
- Obfuscation size: 31 GB

$\Longrightarrow$ **it's almost nearly practical**

Challenge announced at CRYPTO 2014 rump session:

Page 14 from `http://crypto.2014.rump.cr.yp.to/bca480a4e7fcdaf5bfa9dec75ff890c8.pdf`:

Code is available: `https://github.com/amaloz/ind-obfuscation`

ePrint version should be up at some point

For the cryptanalysts in the audience: We have an obfuscated 14-bit point function on Dropbox[1] — learn the point and you win!

[1] `https://www.dropbox.com/s/85d03o0ny3b1c0c/point-14.circ.obf.60.zip`

3 / 3

Breaking the challenge:

- 14-bit point function,
- "60 bits of security",
- obfuscation size: 25 GB.

Breaking the challenge:

- 14-bit point function,
- "60 bits of security",
- obfuscation size: 25 GB.

Attack component	Real time
Initial procrastination	a few days
First attempt to download challenge (failed)	82 minutes

Breaking the challenge:

- 14-bit point function,
- "60 bits of security",
- obfuscation size: 25 GB.

Attack component	Real time
Initial procrastination	a few days
First attempt to download challenge (failed)	82 minutes
Subsequent procrastination	40 days + 40 nights
Fourth attempt to download challenge (succeeded)	about an hour

Breaking the challenge:

- 14-bit point function,
- "60 bits of security",
- obfuscation size: 25 GB.

Attack component	Real time
Initial procrastination	a few days
First attempt to download challenge (failed)	82 minutes
Subsequent procrastination	40 days + 40 nights
Fourth attempt to download challenge (succeeded)	about an hour
Original program evaluating one input	245 minutes
Original program evaluating all inputs on one PC	(extrapolated) 7.6 years

Breaking the challenge:

- 14-bit point function,
- "60 bits of security",
- obfuscation size: 25 GB.

Attack component	Real time
Initial procrastination	a few days
First attempt to download challenge (failed)	82 minutes
Subsequent procrastination	40 days + 40 nights
Fourth attempt to download challenge (succeeded)	about an hour
Original program evaluating one input	245 minutes
Original program evaluating all inputs on one PC	(extrapolated) 7.6 years
Copying challenge to cluster	about an hour
Our faster program evaluating one input	4.85 minutes

Breaking the challenge:

- 14-bit point function,
- "60 bits of security",
- obfuscation size: 25 GB.

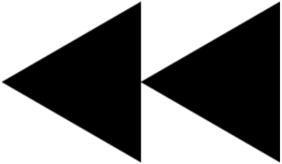
Attack component	Real time
Initial procrastination	a few days
First attempt to download challenge (failed)	82 minutes
Subsequent procrastination	40 days + 40 nights
Fourth attempt to download challenge (succeeded)	about an hour
Original program evaluating one input	245 minutes
Original program evaluating all inputs on one PC	(extrapolated) 7.6 years
Copying challenge to cluster	about an hour
Our faster program evaluating one input	4.85 minutes
First successful break of challenge on 20 PCs	23 hours

Breaking the challenge:

- 14-bit point function,
- “60 bits of security”,
- obfuscation size: 25 GB.

Attack component	Real time
Initial procrastination	a few days
First attempt to download challenge (failed)	82 minutes
Subsequent procrastination	40 days + 40 nights
Fourth attempt to download challenge (succeeded)	about an hour
Original program evaluating one input	245 minutes
Original program evaluating all inputs on one PC	(extrapolated) 7.6 years
Copying challenge to cluster	about an hour
Our faster program evaluating one input	4.85 minutes
First successful break of challenge on 20 PCs	23 hours
Further procrastination (“this is fast enough”)	about half a week
Our faster program evaluating all inputs on 22 PCs	34 minutes
Second successful break of challenge on 21 PCs	19 minutes

What happened?



Restart from the beginning

- (Eurocrypt 2013) Garg, Gentry, and Halevi, *Candidate multilinear maps from ideal lattices.* (GGH)
    - (Crypto 2013) Coron, Lepoint, and Tibouchi, *Practical multilinear maps over the integers.* (CLT)
    - (Eurocrypt 2014) Langlois, Stehlé, Steinfeld, *GGHLite: More Efficient Multilinear Maps from Ideal Lattices.* (GGHLite)

Restart from the beginning

- (Eurocrypt 2013) Garg, Gentry, and Halevi, *Candidate multilinear maps from ideal lattices.* (GGH)
  - (Crypto 2013) Coron, Lepoint, and Tibouchi, *Practical multilinear maps over the integers.* (CLT)
  - (Eurocrypt 2014) Langlois, Stehlé, Steinfeld, *GGHLite: More Efficient Multilinear Maps from Ideal Lattices.* (GGHLite)
- (FOCS 2013) Garg, Gentry, Halevi, Raykova, Sahai, and Waters, *Candidate indistinguishability obfuscation and functional encryption for all circuits.*
  - Several improvements
  - (CCS 2014) Ananth, Gupta, Ishai, and Sahai, *Optimizing obfuscation: avoiding Barrington's theorem.*

Restart from the beginning

- (Eurocrypt 2013) Garg, Gentry, and Halevi, *Candidate multilinear maps from ideal lattices.* (GGH)
  - (Crypto 2013) Coron, Lepoint, and Tibouchi, *Practical multilinear maps over the integers.* (CLT)
  - (Eurocrypt 2014) Langlois, Stehlé, Steinfeld, *GGHLite: More Efficient Multilinear Maps from Ideal Lattices.* (GGHLite)
- (FOCS 2013) Garg, Gentry, Halevi, Raykova, Sahai, and Waters, *Candidate indistinguishability obfuscation and functional encryption for all circuits.*
  - Several improvements
  - (CCS 2014) Ananth, Gupta, Ishai, and Sahai, *Optimizing obfuscation: avoiding Barrington's theorem.*
- Bunch of applications of iO
  - Previously impossible constructions.
  - Reconstructing old things more inefficiently

Restart from the beginning

- (Eurocrypt 2013) Garg, Gentry, and Halevi, *Candidate multilinear maps from ideal lattices.* (GGH)
  - (Crypto 2013) Coron, Lepoint, and Tibouchi, *Practical multilinear maps over the integers.* (CLT)
  - (Eurocrypt 2014) Langlois, Stehlé, Steinfeld, *GGHLite: More Efficient Multilinear Maps from Ideal Lattices.* (GGHLite)
- (FOCS 2013) Garg, Gentry, Halevi, Raykova, Sahai, and Waters, *Candidate indistinguishability obfuscation and functional encryption for all circuits.*
  - Several improvements
  - (CCS 2014) Ananth, Gupta, Ishai, and Sahai, *Optimizing obfuscation: avoiding Barrington's theorem.*
- Bunch of applications of iO
  - Previously impossible constructions.
  - Reconstructing old things more inefficiently *BUT WITH iO!*

The University of Maryland Implementation

- (eprint 2014) Apon, Huang, Katz, and Malozemoff, *Implementing cryptographic program obfuscation.*
- First implementation of candidate obfuscation
- Implement AGIS (CCS 2014) scheme with CLT
- Biggest circuit: 16 bit point function (16 OR gates)

The University of Maryland Implementation

- (eprint 2014) Apon, Huang, Katz, and Malozemoff, *Implementing cryptographic program obfuscation.*
- First implementation of candidate obfuscation
- Implement AGIS (CCS 2014) scheme with CLT
- Biggest circuit: 16 bit point function (16 OR gates)
- Reason? Bad performance!
  - Obfuscation time $\approx$ 7 hours
  - Evaluation time $\approx$ 3 hours
  - Obfuscation size: 31 GB

Point-function obfuscation - Obfuscation

Obfuscation for n-bit point, security parameter λ:

- modulus $q \in \mathbb{N}$ (having $\Theta((\lambda n)^2 \log_2 \lambda)$ bits).
- $2n$ matrices $B_{b,k} \in \mathbb{Z}_q^{(n+2)\times(n+2)}$ for $1 \leq b \leq n$ and $k \in \{0,1\}$,
- $s, t \in \mathbb{Z}_q^{(n+2)}$,
- zero test value $p_{\mathrm{zt}} \in \mathbb{Z}_q$ (Multi-linear map artifact).

Point-function obfuscation - Evaluation

Evaluation for $x = (x[1], x[2], \ldots, x[n]) \in \{0, 1\}^n$

- Compute $A = B_{1,x[1]} B_{2,x[2]} \cdots B_{n,x[n]}$.
- Compute $y(x) = s^\top A t$.
- Compute $y(x) p_{\mathrm{zt}}$ and reduce mod q to the range $[-(q-1)/2, (q-1)/2]$.
- Multiply the remainder by $2^{2\lambda+11}$, divide by q, and round to the nearest integer.
- Output 0 if result is 0; output 1 otherwise.

Performance & Security

Performance:

- Experimentally ($n = 14$): $\approx 2^{49}$ cycles / evaluation
- Theoretically:
  - $(n-1)(n+2)^3$ multiplications of integers $> (q-1)^n$ (on average) to compute A.
  - Total complexity $n^{7+o(1)}$ (assuming schoolbook multiplication).

Performance & Security

Performance:

- Experimentally ($n = 14$): $\approx 2^{49}$ cycles / evaluation
- Theoretically:
  - $(n-1)(n+2)^3$ multiplications of integers $> (q-1)^n$ (on average) to compute A.
  - Total complexity $n^{7+o(1)}$ (assuming schoolbook multiplication).

Security:

- Parameters chosen for 60 bit security of multi-linear map.
- "Exhaustive search too slow" ($> 2^{60}$ cycles for $n = 14$).

Our attack

Our attack

1. Speed up evaluation
   1.1 Intermediate reductions mod q
   1.2 Only matrix-vector products
2. Speed up exhaustive search
   2.1 Reuse intermediate results
   2.2 Meet-in-the-middle

Intermediate reductions mod q

- Integers grow up to $(n+2)^{n-1}(q-1)^n$; typically larger than $(q-1)^n$.
- Multiplication essentially linear in $\#$bits ($b^{1+o(1)}$).
- $y(x)p_{\mathrm{zt}}$ is in $\mathbb{Z}_q$.

Intermediate reductions mod q

- Integers grow up to $(n+2)^{n-1}(q-1)^n$; typically larger than $(q-1)^n$.
- Multiplication essentially linear in #bits ($b^{1+o(1)}$).
- $y(x)p_{\mathrm{zt}}$ is in $\mathbb{Z}_q$.

Improvement:

- Reduce mod q after every vector-vector product.
- Inputs to mult $< q-1$.
- Reduce costs by factor n:

$$n^{7+o(1)} \Rightarrow n^{6+o(1)}$$

Only matrix-vector products

Step 1 of eval: Compute $A = B_{1,x[1]} B_{2,x[2]} \cdots B_{n,x[n]}$.
Step 2 of eval: Compute $y(x) = s^\top A t$.

Only matrix-vector products

Step 1 of eval: Compute $A = B_{1,x[1]} B_{2,x[2]} \cdots B_{n,x[n]}$.
Step 2 of eval: Compute $y(x) = s^\top A t$.

- Matrix-matrix products are unnecessary!
- $y(x) = \left(\cdots\left(\left(s^\top B_{1,x[1]}\right) B_{2,x[2]}\right) \cdots B_{n,x[n]}\right) t$
- $(n-1)(n+2)^3$ multiplications $\Rightarrow (n-1)(n+2)^2$ (omitting vector-vector)

$$n^{6+o(1)} \Rightarrow n^{5+o(1)}$$

From single evaluation to exhaustive search

So far we reduced time for one evaluation

$$n^{7+o(1)} \Rightarrow n^{5+o(1)},$$

dominated by $(n+1)^2$ dot products mod q.
This means, exhaustive search takes

$$(n+1)^2 2^n$$

dot products mod q.

Reuse intermediate results

- $y(\mathbf{0}) = \left(\cdots\left((s^\top B_{1,0})B_{2,0}\right)\cdots B_{n-1,0}\right)\cdots B_{n,\color{red}{0}}\, t$
- $y(\mathbf{1}) = \left(\cdots\left((s^\top B_{1,0})B_{2,0}\right)\cdots B_{n-1,0}\right)\cdots B_{n,\color{red}{1}}\, t$

Reuse intermediate results

- $y(\mathbf{0}) = \big(\cdots\big((s^\top B_{1,0})B_{2,0}\big)\cdots B_{n-1,0}\big)\cdots B_{n,\textcolor{red}{0}}t$
- $y(\mathbf{1}) = \big(\cdots\big((s^\top B_{1,0})B_{2,0}\big)\cdots B_{n-1,0}\big)\cdots B_{n,\textcolor{red}{1}}t$
- Generally, trying all inputs in order, average cost to update result: 2 vector-matrix products.

Improvement:

$$(n+1)^2 2^n \Rightarrow 2(n+1)2^n \text{ dot products mod } q$$

Meet-in-the-middle

- Split computation into two halves

$$y(x) = (s^\top B_{1,x[1]} \cdots B_{\ell,x[\ell]})(B_{\ell+1,x[\ell+1]} \cdots B_{n,x[n]} t).$$

Meet-in-the-middle

- Split computation into two halves

$$y(x) = (s^\top B_{1,x[1]} \cdots B_{\ell,x[\ell]})(B_{\ell+1,x[\ell+1]} \cdots B_{n,x[n]} t).$$

- precompute a table of "left" products

$$L[x[1], \ldots, x[\ell]] = sB_{1,x[1]} \cdots B_{\ell,x[\ell]}$$

for all 2^ℓ choices of $(x[1], \ldots, x[\ell])$

Meet-in-the-middle

- Split computation into two halves

$$y(x) = (s^\top B_{1,x[1]} \cdots B_{\ell,x[\ell]})(B_{\ell+1,x[\ell+1]} \cdots B_{n,x[n]} t).$$

- precompute a table of "left" products

$$L[x[1], \ldots, x[\ell]] = sB_{1,x[1]} \cdots B_{\ell,x[\ell]}$$

  for all 2^ℓ choices of $(x[1], \ldots, x[\ell])$
- for each choice of $(x[\ell+1], \ldots, x[n])$, compute "right" product

$$R[x[\ell+1], \ldots, x[n]] = B_{\ell+1,x[\ell+1]} \cdots B_{n,x[n]} t,$$

  and multiply each element of the L table by this vector.

Meet-in-the-middle

- Split computation into two halves

$$y(x) = (s^\top B_{1,x[1]} \cdots B_{\ell,x[\ell]})(B_{\ell+1,x[\ell+1]} \cdots B_{n,x[n]} t).$$

- precompute a table of "left" products

$$L[x[1], \ldots, x[\ell]] = sB_{1,x[1]} \cdots B_{\ell,x[\ell]}$$

  for all 2^ℓ choices of $(x[1], \ldots, x[\ell])$
- for each choice of $(x[\ell+1], \ldots, x[n])$, compute "right" product

$$R[x[\ell+1], \ldots, x[n]] = B_{\ell+1,x[\ell+1]} \cdots B_{n,x[n]} t,$$

  and multiply each element of the L table by this vector.

Improvement for $\ell = n/2$ (assuming n is even):

$$2(n+1)2^n \Rightarrow 4(n+2)(2^{n/2}-1) + 2^n \text{ dot products mod } q$$

Summing up

We reduced time for exhaustive search from

$$n^{7+o(1)}2^n \Rightarrow n^{3+o(1)}2^n.$$

On one PC this takes 444.2 minutes. (Original program, estimated $= 7.6$ years)
On 22 PCs: 29.5 minutes.

Conclusion

- First two improvements speed up obfuscation scheme (though trivial).
- Reuse and meet-in-the-middle vulnerabilities inherent to obfuscation using matrix-branching (always at least factor n^2 speed-up).
- See paper for generalizations and further asymptotic speedups

Conclusion

- First two improvements speed up obfuscation scheme (though trivial).
- Reuse and meet-in-the-middle vulnerabilities inherent to obfuscation using matrix-branching (always at least factor n^2 speed-up).
- See paper for generalizations and further asymptotic speedups

Simply use a hash function for password-hashing!

Conclusion

- First two improvements speed up obfuscation scheme (though trivial).
- Reuse and meet-in-the-middle vulnerabilities inherent to obfuscation using matrix-branching (always at least factor n^2 speed-up).
- See paper for generalizations and further asymptotic speedups

Simply use a hash function for password-hashing!
Thank you.