Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available.

2016 © Microsoft Corporation. All rights reserved.
Cryptographic Program Obfuscation: Current Capabilities & Challenges

Yuriy Polyakov
polyakov@njit.edu
MSR May 1 2018
NJIT Cybersecurity Research Center

- Newly founded center
  - 5 tenure-track faculty
  - 1 research professor, 2 scientists
  - ~10 PhD students

- Expertise in:
  - Encrypted Computing
  - Mobile/Android Security
  - Embedded system security
  - Applied Cryptography
PALISADE Lattice Cryptography Library

- Project-based Development since 2014
  - Next generation of DARPA PROCEED SIPHER project
  - Cryptographic program obfuscation (DARPA Safeware)
  - Homomorphic Encryption for statistical analysis (Sloan, IARPA)
  - Proxy Re-Encryption for Pub/Sub systems (Simons, NSA)
  - HE backend for Secure Programming in Julia (IARPA)

- Implementation Partners and Collaborators
  - Academia: MIT, UCSD, WPI, NUS, Sabanci U
  - Industry: Raytheon (BBN), IBM Research, Lucent, Vencore Labs, Galois, Two Six Labs

- BSD 2-clause license

- Cross-Platform Support

Yuriy Polyakov polyakov@njit.edu
DARPA SAFEWARE

• DARPA
  – Founded in 1958 (after Sputnik 1 launch)
  – Past projects: ARPANET, wavelets, onion routing, etc.
  – DARPA PROCEED program (BGV11, GHS12, BGH12, Brakerski12, etc.)

• DARPA SAFEWARE
  – Started in 2015
  – PI groups: MIT, IBM Research, NJIT, Columbia, UCSD, Berkeley, Yale, Stanford, John Hopkins, Raytheon BBN, Vencore Labs (ACS), Galois, etc.
Obfuscation?

- What happens when you want to protect the computation???

- Example:
  - You do extensive work to train a neural net.
    - Years and years of effort to design the best classifier ever.
  
- Once you deploy this classifier, anyone could steal it and reverse-engineer it.
  - FHE won’t protect you.
Obfuscation

- Can we design programs that can’t be reverse engineered?
  - Compiled software can be decompiled.
  - Obfuscation techniques are currently mostly heuristic.

- Can we design software to be a cryptographic black box?
  - Can we guarantee that reverse engineering a program is at least as hard as exhaustively exploring input-output map?
Cryptographic Obfuscation

- Who cares? So what?
- Hardware can be captured / stolen / purchased.
  - Hardware is very cheap.
  - Software is shockingly expensive.
- Do we want adversaries to be able to reverse engineer classifiers, autonomous systems, etc...
  - Industry makes tremendous efforts to generate new IP with extremely expensive human resources.

Yuriy Polyakov polyakov@njit.edu
Obfuscation Vision

- Take arbitrary program, and obfuscate it into byte code.
  - Ex: Java Program converted into Obfuscated Java Byte Code

- Deploy byte code on a VM
  - Ex: Run Obfuscated Java Byte code on JVM
Motivating Applications

- Linear classifiers: Automatic Target Recognition systems
- Non-linear classifiers
- Linear and non-linear filters
- Supervisory control and data acquisition (SCADA) control systems
Theory of General-Purpose CPO: Milestones

- Barak et al. show that general-purpose VBB obfuscation is unachievable and introduce the concept of indistinguishability obfuscation [IO] (2001) [1]
- Garg, Gentry, and Halevi design the graded multi-linear map encoding from ideal lattices for IO (2013) [2]
- Coron, Lepoint, and Tibouchi propose the graded multi-linear map encoding over integers for IO (2013) [3]
- Gentry, Gorbunov, and Halevi propose graph-induced multilinear map encoding based on lattices (2015) [4]
- All these constructions are currently BROKEN from the perspective of general-purpose IO
Implementation results for CPO

- Apon et al. build a branching program (BP) to obfuscate point functions [5]. A 14-bit point function takes 9 hours to obfuscate (60-bit security), program size is 31 GB, and evaluation runtime is 3.3 hours.

- Lewi et al. (CCS ’16) obfuscate an 80-bit point function with 80 bits of security [6]. Obfuscation time: 3.3 hours, program size: 8.3 GB, evaluation time: 180 seconds.

- Halevi et al. (CCS ’17) obfuscate oblivious read-once BPs (nondeterministic finite automata of at most 80 bits with about 100 states) [7]. Obfuscation time: 23 days. Program size: 9 TB. Evaluation time: 25 min.

- Our group (S&P ’18) obfuscates 32- and 64-bit conjunction programs [8]. For the 32-bit conjunction function, obfuscation time: 6.2 min, program size: 5.85 GB, evaluation time: 32 ms. For the 64-bit program, obfuscation time: 6.7 hrs, program size: 748 GB, evaluation time: 2.45 s.
Conjunction Obfuscation: KeyGen and Obfuscation

Parameters:
- $q = 3453472546546...$
- $n = 8192$

Pattern: 1?01

KeyGen (Run Offline)

"Keys"

Obfuscator (Run Offline)

Obfuscated Program

Yuriy Polyakov polyakov@njit.edu
Conjunction Obfuscation Evaluation

Input: 0100

Obfuscated Program

Evaluator (Run Online)

Output: 0
(Input doesn't match pattern in this example.)

Obfuscated Input Pattern: 1?01

Yuriy Polyakov polyakov@njit.edu
Obfuscated Application
Biometric Matching

- Goal: Obfuscate Program to match faces, irises, fingerprints to a "signature" of interest.

- Practicality is (one of) the research challenges we are addressing over coming Phase(s).
  - Can we speed this up enough to make it real???
Obfuscation

Input Pattern: 1?01
We can represent program as a Finite State Machine (FSM)
(we can call this a Moore machine.)

---

Yuriy Polyakov polyakov@njit.edu
Obfuscation

- Input bits drive state transitions
- Represent state transitions as multiplications of ring elements.
  - We encode the ring elements using lattice trapdoor functions.
  - Encoded as matrices of ring elements.
- We return 1 if the chosen path’s product matches the correct path’s product.
Program

S matrices
0,1  0,2  0,3
1,1  1,2  1,3

R matrices
0,1  0,2  0,3
1,1  1,2  1,3

Matrices of Ring Elements encode state transitions

Yuriy Polyakov polyakov@njit.edu
Program Evaluation - 1

Input:

```
| 1 | 0 | 0 | 1 |
```

S matrices:

```
| 0,1 | 0,2 | 0,3 | 0,L |
| 1,1 | 1,2 | 1,3 | 1,L |
| 0,1 | 0,2 | 0,3 | 0,L |
| 1,1 | 1,2 | 1,3 | 1,L |
```

R matrices:

```
| 0,1 | 0,2 | 0,3 | 0,L |
| 1,1 | 1,2 | 1,3 | 1,L |
```

Matrices of Ring Elements that are ring encodings

Yuriy Polyakov polyakov@njit.edu
Select Arrays of matrices that correspond to Input

1 0 0

_______

1,1
0,2
0,3

_______

1,1
0,2
0,3

_______

1,1
1,L
Program Evaluation - 3

Compute Products of Arrays of Matrices

1,1  0,2  0,3  1,L

Product of Array  Prod

1,1  0,2  0,3  1,L

Product of Array  Prod

Yuriy Polyakov polyakov@njit.edu
If the result is "small", then program returns 1.
How practical is CPO?

- When we started at the end of 2015, the obfuscated 32-bit program had the following estimated performance:
  - Key generation time: 3 years
  - Key size: 15 PB
  - Program size: 600 TB
  - Obfuscation time: 70 hrs
  - Evaluation time: 9 hrs
How practical is CPO?

- Our results as of September 2017
  - Key generation time: 100 ms
  - Key size: 1 MB
  - Program size: 5.85 GB
  - Obfuscation time: 6.2 min
  - Evaluation time: 32 ms
How practical is CPO?

- When we started at the end of 2015, the obfuscated 32-bit program had the following estimated performance:
  - Key generation time: 3 years
  - Key size: 15 PB
  - Program size: 600 TB
  - Obfuscation time: 70 hrs
  - Evaluation time: 9 hrs
Summary of improvements

• Dramatically improved lattice trapdoor construction and sampling algorithms. We used this lattice trapdoor construction to implement GPV signature, IBE, CP-ABE, and KP-ABE [9-11].
• Generalized alphabet for directed encoding
• More careful correctness and security constraints
• Efficient matrix arithmetic
• Efficient polynomial arithmetic
• Efficient integer arithmetic and integer sampling
• Parallelization at multiple levels
• and more....
Still a lot of practicality questions

- How to efficiently implement more complex functions?
- Is it possible to build general-purpose obfuscation constructions that cannot be broken?
- Are there other major algorithmic improvements that can reduce the obfuscated program size requirements?
What can we do?

- Explore other approaches that achieve similar results in practice
  - Token-based approaches (interactive schemes) that limit the number of I/O queries
  - We’ve developed several prototypes for token-based obfuscation of linear classifiers, conjunctions, and permutation branching programs
- Build CPO for special functions, e.g., evasive functions, based on the results for conjunction obfuscation. This approach is secure under standard assumptions, namely Learning with Errors.
Questions?
Yuriy Polyakov
polyakov@njit.edu