Implementing BP-Obfuscation Using Graph-Induced Graded Encoding

Shai Halevi
Tzipora Halevi
Victor Shoup
Noah Stephens-Davidowitz

https://eprint.iacr.org/2017/104

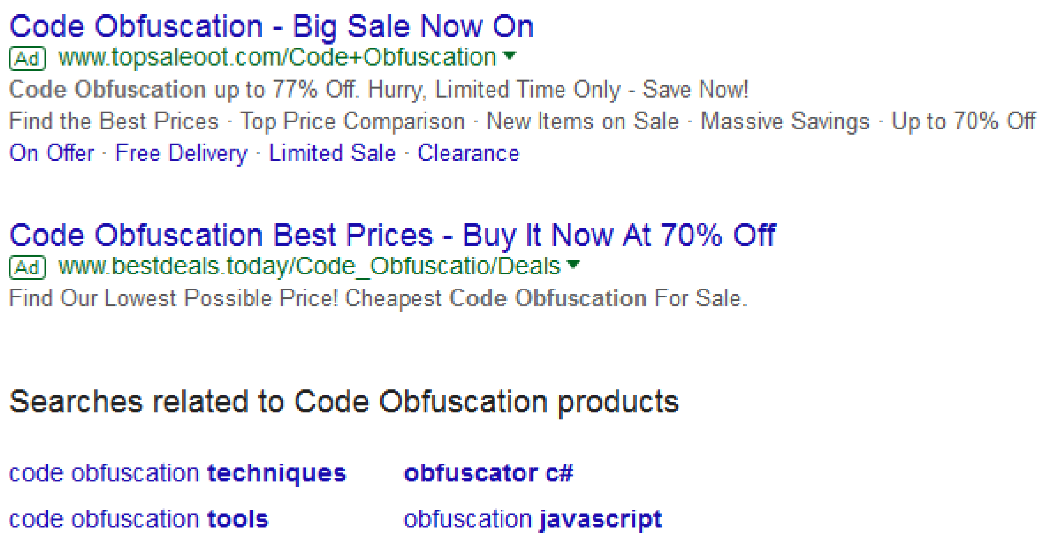
Supported by the Defense Advanced Research Projects Agency (DARPA) and Army Research Office (ARO) under Contract No. W911NF-15-C-0236.

Program Obfuscation

- Make program "unintelligible"
  - Hide inner workings, only I/O should be "visible"
- Enable hiding secrets in software
  - E.g. cryptographic key, or an algorithm
- We seek an obfuscating compiler:
  - Arbitrary program in, obfuscated program out
  - Without changing the functionality
  - At most polynomial slowdown

Obfuscation is Useful

- Commercially available ad-hoc obfuscation



  - Heuristic, trying to make reverse-engineering harder
  - Can always be broken with “enough debugging”
- Can we get “crypto-strength” obfuscation?

Cryptographic Obfuscation

- 1st plausible construction in [GGHRSW'13]
  - Several others since then
- Constructions have a "core component" that obfuscates "somewhat simple" programs
  - E.g., "branching programs" (BPs)
- Then a transformation that extends it to general programs
  - Using other tools (e.g., FHE, NIZK, RE, etc.)

How to Obfuscate?

- Main tool is "graded encoding" [GGH'13]
  - Like homomorphic encryption, values can be hidden by "encoding", but still manipulated
  - Main difference: can see if the encoded value is 0
- High-level idea: run program on encoded values, check at the end if the result is zero
  - Main problem: hiding whether or not any two intermediate values are the same
  - Use randomization techniques for that

CRYPTOGRAPHIC OBFUSCATION CHALLENGES

- Security is poorly understood
- Current-day graded encoding is very costly
  - Other components make “core obfuscator” more costly still
- Previous implementation attempts:
  - [AHKM’14]: 14-bit point function
  - [LMA+’16] (5Gen): 80+ bit point function
    - More accurately 20+ nibbles
  - Note: point functions can be obfuscated much faster using special-purpose constructions

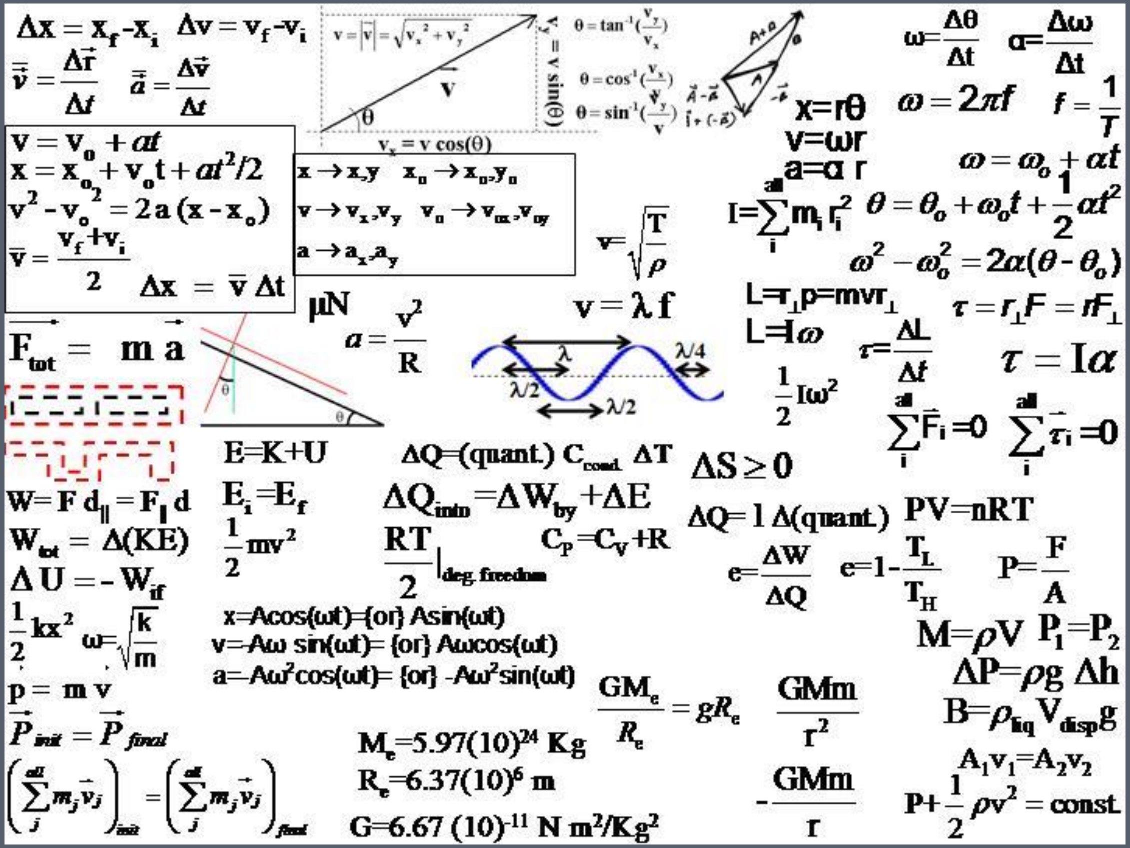
Our Work

- Obfuscate “read once branching programs”
  - Aka nondeterministic finite automata (NFA)
- Can handle ~100 states & upto 80-bit inputs
  - More accurately, 20 nibbles
- Can obfuscate some non-trivial functions
  - E.g., Substring/superstring/fuzzy match
- Still not enough for the “somewhat simple functions” that we would like to handle

Our Work

- Using the “graph-induced” graded encodings scheme of Gentry et al. [GGH’15]
  - Previous implementations used the encoding scheme of Coron et al. [CLT’13]
  - GGH15 seems better for NFAs with many states
- For performance reasons, could not implement one of the steps in [GGH’15]
  - Namely, the “bundling factors”

  ➔ implementation is only safe when used to obfuscate read-once BPs, not arbitrary BPs



SOME DETAILS

don’t worry, only three slides

Obfuscating BPs/NFAs

- Graphs, represented by transition matrices
  - Need to "hide" matrices, but allow them to be multiplied and compared to zero
- Begin by randomizing these matrices
  - Mainly Kilian-style randomization: $M_1 \times M_2 \times M_3 \to (M_1 R_1) \times (R_1^{-1} M_2 R_2) \times (R_2^{-1} M_3)$
- Apply graded encoding to randomized matrices
- Can multiply encoded matrices, check for zero
  - But cannot "see" the original matrices

"Graph-induced" Graded Encoding

- Parametrized by a chain of matrices A_i

$$A_0 \xrightarrow{M_1} A_1 \xrightarrow{M_2} A_2 \xrightarrow{M_3} \dots \xrightarrow{M_n} A_n$$

- We encode "plaintext matrices" wrt edges
- Encoding of M_i wrt $A_{i-1} \to A_i$ is a low-norm matrix C_i s.t., $\boldsymbol{A_{i-1}C_i = M_iA_i +}$ **small-error**
  - The "hard part" is finding such a low-norm C_i

"Graph-induced" Graded Encoding

- Parametrized by a chain of matrices A_i

$$A_0 \xrightarrow{M_1} A_1 \xrightarrow{M_2} A_2 \xrightarrow{M_3} \dots \xrightarrow{M_n} A_n$$

- We encode "plaintext matrices" wrt edges
- Encoding of M_i wrt $A_{i-1} \to A_i$ is a low-norm matrix C_i s.t., $\boldsymbol{A_{i-1} C_i = M_i A_i} +$ **small-error**
  - The "hard part" is finding such a low-norm C_i
- It follows that $A_0 \prod_i C_i = (\prod_i M_i) A_n$ + small-error
  - At least when the M_i's themselves are small
- To test if $\prod_i M_i = 0$, check the size of $A_0 \prod_i C_i$

Our Main Optimizations

- Finding a small solution C for $AC = B$:
  - Variant of trapdoor-sampling from [MP'12]
  - A new high-dimensional Gaussian lattice sampling
  - Working with integers in CRT representation
- Optimizing multiplication of very large matrices
  - Each matrix takes more than 18Gb to write down
- Many lower-level optimizations
  - Stash to reduce the number of samples, multi-threading strategies, memory-saving methods, …

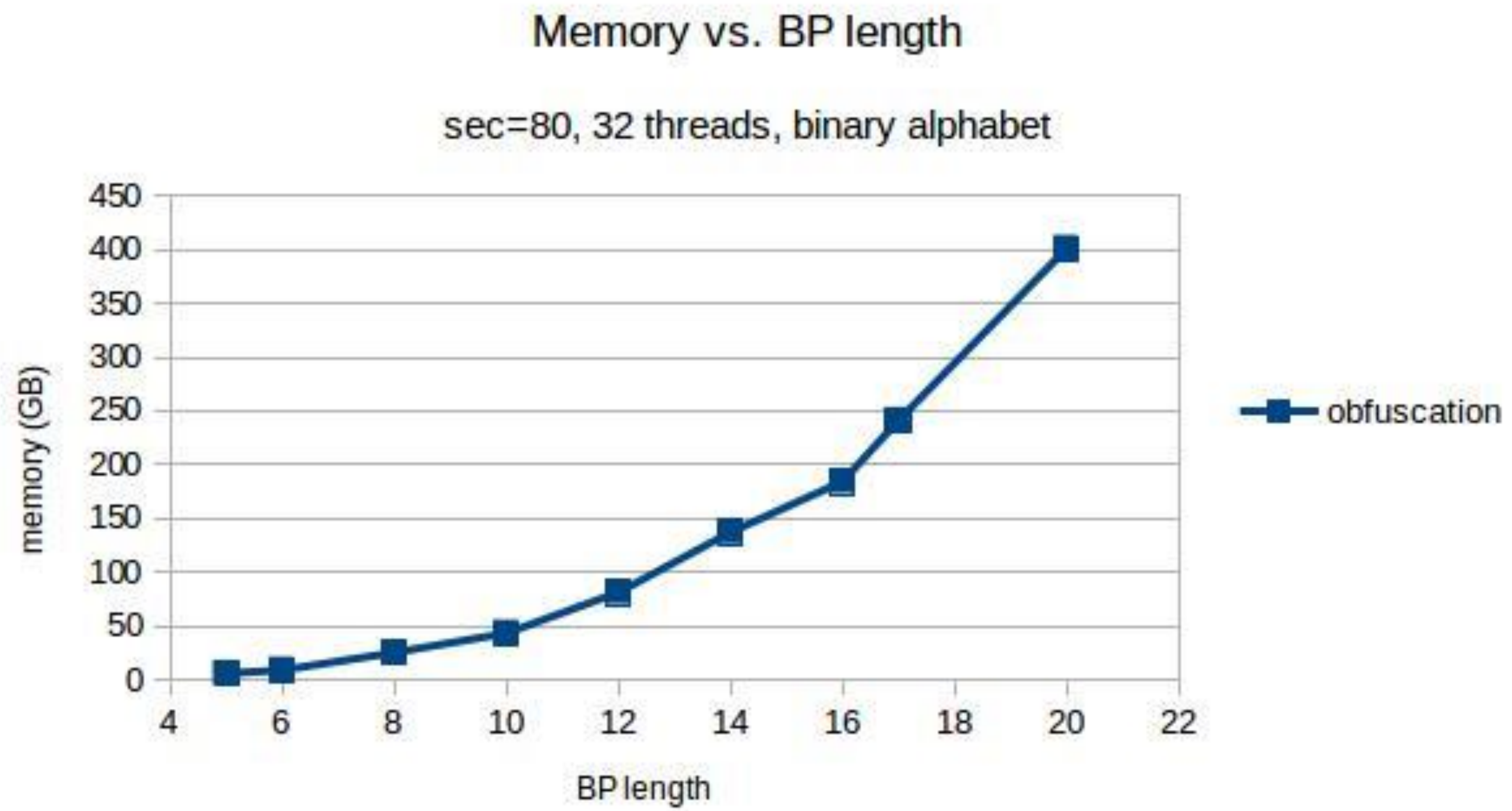
Some Performance Numbers

L	m	Initialization	Obfuscation	Evaluation
Intel Xeon CPU,E5-2698 v3:				
5	3352	66.61	249.80	5.81
6	3932	135.33	503.01	13.03
8	5621	603.06	1865.67	56.61
10	6730	1382.59	4084.14	125.39
12	8339	3207.72	8947.79	300.32
14	9923	7748.91	18469.30	621.48
16	10925	11475.60	38926.50	949.41
17	11928	16953.30	44027.80	1352.48
18	12403	20700.00	out-of-RAM	
4 x 16-core Xeon CPUs:				
17	11928	16523.7	84542.3	646.46
19	13564	36272.9	182001.4	1139.36
20	14145	46996.8	243525.6	1514.26

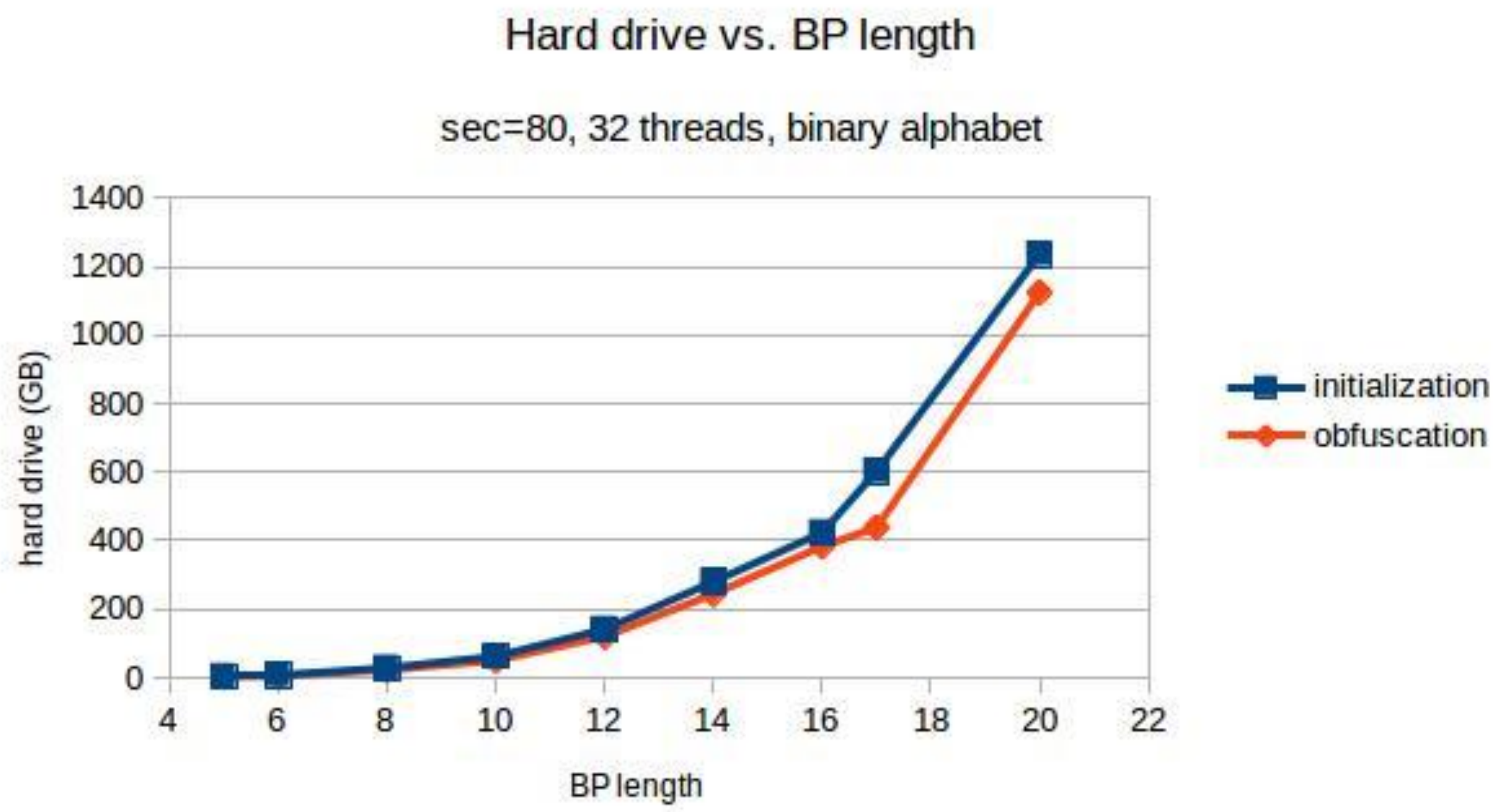
68 hours

100 states, security=80, binary alphabet. L=input length, m=dimension

Some Performance Numbers



Some Performance Numbers



Some Performance Numbers

- When using “nibbles” rather than bits for input:
  - Obfuscation time, disk usage, 8x increase
  - Everything else remains the same
- To handle BP of length 20 with input nibbles:
  - Init: 13hrs, obfuscate: 23 days, Eval: 25mins
  - RAM: 400GB
  - Disk space: ~10TB

Conclusions

- Cryptographic "general-purpose obfuscation" is barely feasible
  - Can handle some non-trivial functions
  - With inputs up to 20 characters (=80 bits)
- A new generation of constructions is now emerging [Lin'16,…]
  - Security is somewhat better understood
  - Practical performance still unknown
    - Could be better than previous constructions, or worse

Questions?

REFERENCES

- **[MP'12]** Micciancio and C. Peikert. *Trapdoors for lattices: Simpler, tighter, faster, smaller.* Eurocrypt 2012
- **[GGH'13]** Garg, Gentry, Halevi. *Candidate Multilinear Maps from Ideal Lattices*. Eurocrypt 2013
- **[CLT'13]** Coron, Lepoint, Tibouchi. *Practical multilinear maps over the integers.* CRYPTO 2013
- **[GGHRSW'13]** Garg, Gentry, Halevi, Raykova, Sahai, Waters. *Candidate indistinguishability obfuscation and functional encryption for all circuits.* SIAM J. Comput., 45(3):882-929, 2016.
- **[AHKM'14]** Apon, Huang, Katz, Malozemo. *Implementing cryptographic program obfuscation.* http://eprint.iacr.org/ 2014/779
- **[GGH'15]** Gentry, Gorbunov, Halevi. *Graph-induced multilinear maps from lattices.* TCC 2015
- **[LMA+'16]** Lewi, Malozemo, Apon, Carmer, Foltzer, Wagner, Archer, Boneh, Katz, Raykova. 5Gen: *A framework for prototyping applications using multilinear maps and matrix branching programs.* CCS 2016
- **[Lin'16]** Indistinguishability obfuscation from constant-degree ideal graded encoding, Eurocrypt 2016