Obfuscating Straight-line Arithmetic Programs

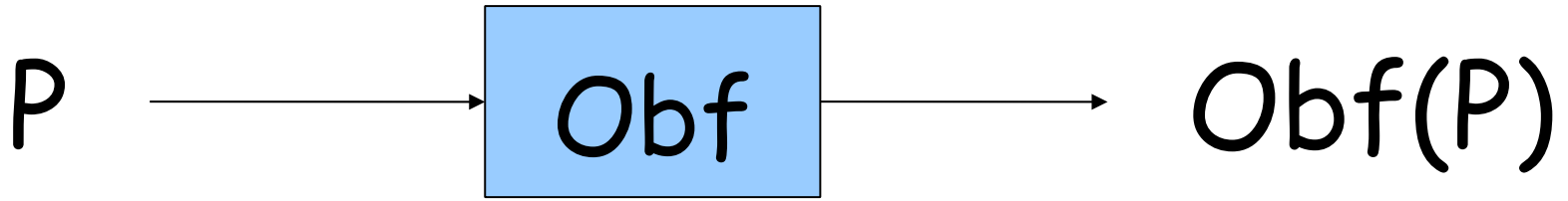
Ramarathnam Venkatesan
Microsoft Research (Redmond and India)

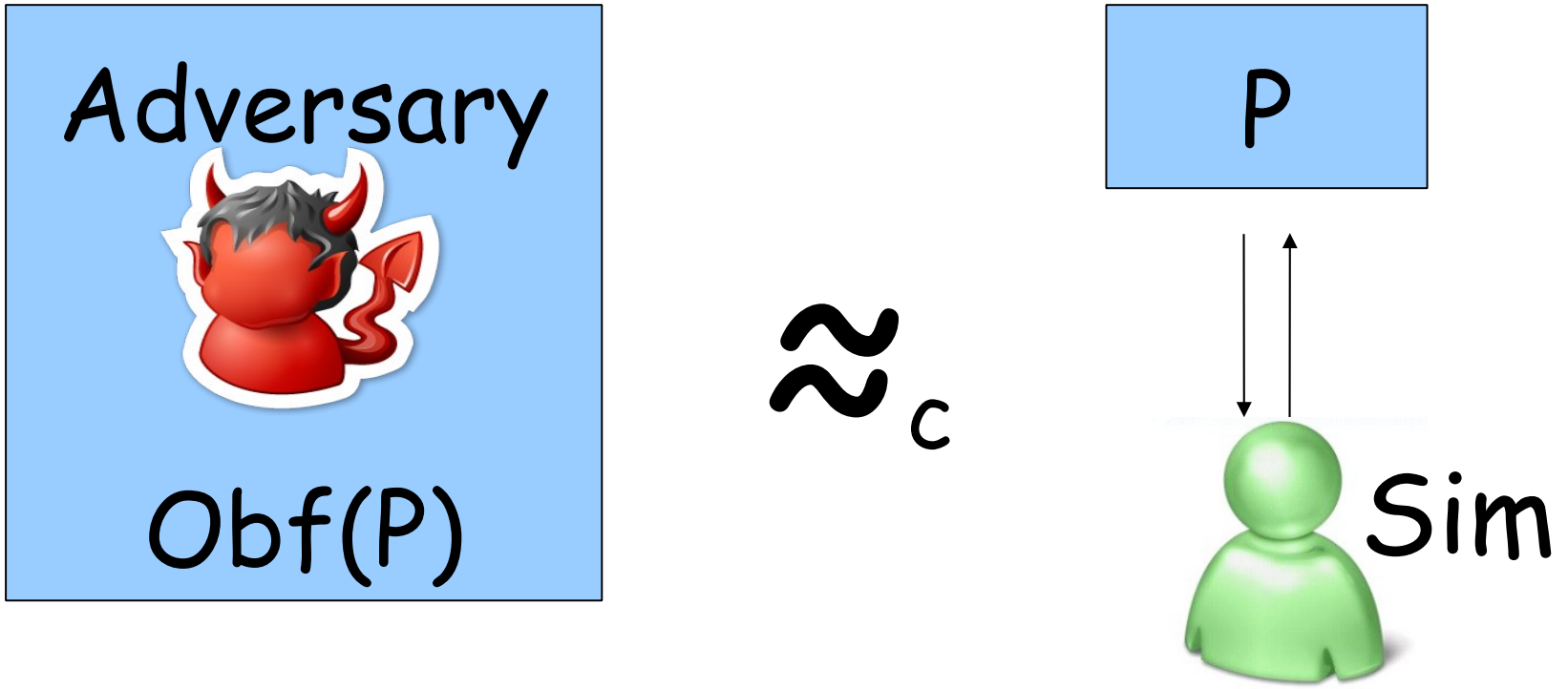
Srivatsan Narayanan*
IIT Madras, CMU

Ananth Raghunathan*
IIT-Madras, Stanford

*Work done as interns at Microsoft Research India, Bangalore

What is obfuscation?





Properties of Obf()

- Polynomial time in |P|
- At most polynomial blowup in the program size
  - |Obf(P)| <= poly(|P|)
- Every adversary with access to Obf(P) can be simulated by a good program Sim with only oracle access (i.e., functionality) to P

Why obfusction?

- Holy grail of cryptography
- Prevents "cracking" of commercial software
  - Software protection for proprietary software
  - Prevents reverse-engineering
- Convert any private-key crypto-system into a public-key crypto-system
- Allow delegation of computation (like digital signatures) to untrusted parties

Previous results

- [Barak et al '01] Impossible to obfuscate all functions (there are inherently unobfuscatable functions)
  - These functions were however not natural for cryptographic applications
- [Goldwasser-Kalai '05] Stronger notion of obfuscation (with auxiliary input)
  - Showed impossibility for "natural" functions – pseudo random functions, probabilistic digital signatures schemes

Then why discuss obfuscation?

- These models are too strong. Impossibility results are expected
- Positive results with weaker models
  - [Lynn et al '04] Access control in the random oracle
  - [Wee STOC '05] Point-functions assuming trapdoor one-way permutations
  - [Hohenberger et al TCC '07] Obfuscating re-encryption
  - [Goldwasser et al CRYPTO '08] One-time programs with hardware support

Overcoming negative results

- Weaker notions of security
  - Limit security requirement over smaller class of functions (like in [Wee '05] and [Hohenberger et al '07])
- Additional support for obfuscator
  - Secure, tamper-proof hardware
  - One-time hardware ([Goldwasser et al '08])

Secure Tamper-proof Hardware

- Sufficiency [Goldreich-Ostrovsky '96]
  - Tamper-proof stateful CPU with encryption can obfuscate any program
  - Overkill! Cannot make entire CPU from tamper-proof hardware
- More practical notion of tamper-proof hardware
  - Stateless hardware token (eg. USB stick)
  - Small secret stored in hardware token (independent of program)
  - Can this help obfuscate programs obfuscation?

How to realize this?

- Computer vendor programs CPU to interact with a secure hardware token T via USB (say)
- When running a obfuscated program
  - Naïve execution on CPU will not work
  - Small number of interactions between CPU and USB stick (modeled as oracle calls)
  - Sufficient to successfully execute obfuscated program
  - If USB stick is secure software program cannot be cracked or reverse-engineered

Previous Work

- Obfuscating circuits [Goyal-V. '09]
  - Notion of Obfuscation-complete oracle
  - Single circuit which if stored in secure tamper proof hardware can obfuscate all circuits
  - Overhead of only log(|C|) where |C| is the size of the circuit
    - Better than the [Goldreich-Ostrovsky] construction by O(log^2(|C|))
  - Can we extend this to classes beyond circuits? Yes!

Straight-line Arithmetic Programs

- Let **F** be a field and X={x1,x2,…,xm} be a set of intermediates. P(X,V,C,S) is an arithmetic straight-line program over **K**=**F**(x1,…,xm) if
  - S={s1,s2,…,sk} ⊂ **F**, V=v1,v2,…,vl, V∩K=∅ (X=set of inputs, V=set of variables, S=scalars)
  - We define a computation sequence on S∪X∪{v1,v2,…,vl}
  - F ∈ **F**[X1,X2,…,Xm], a polynomial, is said to represent or compute the straight-line program

Obfuscation Model

- OracleGen(1^k) produces (Obf, S)
- T is a tamper-proof hardware with the secret S embedded inside.
- k is the security parameter
- 3 properties
  - Encrypted functionality
  - Polynomial Slowdown and Efficient Hardware
  - Security

Obfuscation Model (contd)

- Obf takes input program P and oracle access to T and outputs P'=Obf^T(P)
- Obfuscated program P' uses oracle access (with small number of oracle queries) upon input x, and enables T to compute P(x) in polynomial time
- T then outputs Enc(P(x)) where Enc() is some encryption function

Why encryption?

- Old result by Kaltofen
- [Kal 85] Straight-line arithmetic programs are "learnable"
- In other words, there is an algorithm that given $f \in \mathbf{F}[X1,X2,\ldots,Xm]$ outputs irreducible h_i each given by a straight-line program of polynomially bounded length, and e_i such that

$f=\Pi(h_i)^{(e_i)}$

- Therefore, given a polynomially number of evaluations of f, we can "reconstruct" f as a product of irreducible factors
- Previous work by [Ostrovsky-Skeith '05] on obfuscation with encryption

Polynomial Slowdown and Efficient Hardware

- There exist polynomials p() and q() such that for sufficiently large k and |P| we have

$|Obf(P)| \leq p(|P|, k)$

$|S| \leq q(k)$

- In other words, the size of the output obfuscation and the secret are polynomial in both the input program size and the security paramater

Security

- For every PPT adversary A that takes input Obf(P) and is able to output P' there exists a negligible function e() such that for every polynomial p, we have

for all x,

$$\Pr[P'(x)=P(x) \mid A(Obf(P),1^k)=P'] \leq e(k)$$

Hard-to-factor polynomials

- Shamir's work back in 1993
- Algebraic form
  - Multivariate polynomial where each coefficient is a rational function of params (a,b,...)
- Algebraic collection (for n=pq)
  - Set of all polynomials (mod n) is the set of all polynomials in variables x,y,z,... with numeric coefficients that are obtained by substituting params a,b,... with all values in [0,n)
- Induces a distribution on the class of all multivariate polynomials in x,y,z

Main Result in [Sha '93]

- Pick two polynomials P and Q from a class C.
  - i.e., two assignments to params a,b,...
- Set F=PQ (mod n)
- Recovering two polynomials S and T such that ST=F (mod n) is at least as hard as factoring
- Can be provably reduced to factoring n
- Can we use this provable reduction to obfuscate polynomials? (notice, F is obfuscating P and Q)

Outline of construction

- OracleGen (1^k) chooses sufficiently many primes p_i and q_i. n_i=p_i q_i. This comprises secret S
- Obf() on input f does the foll. for each i
  - Computes f (mod p)
  - Chooses random g (mod q) and uses chinese remainder theorem to create P (mod n) which is

= f (mod p) and = g (mod q)

  - Chooses a random Q (mod n) from same distribution as P
  - Publish <PQ, P+Q> and n. However, p, q, are private

Outline of construction (contd)

- How to execute the obfuscated program?
  - Adversary A on input x evaluates a=P(x) Q(x) (mod n) and b=P(x)+Q(x) (mod n)
  - Passes a and b to token t
  - Token computes a'=a mod p and b'=b mod p
  - It solves for y and z such that

yz=a' (mod p) and y+z=a' (mod p)

Intractable for p not prime, but easy to do for primes of the form 4t+1 (extracting square roots)

  - Repeat for each p_i and output (using CRT) Enc(f(x)) (mod $\prod$p_i)
  - If i is suitably large, this is equal to f(x)

Proof outline

- Encrypted functionality
- Polynomial Slowdown and Efficient Hardware
  - Extract square root in polynomial time for primes of the form 4n+3
  - deg(P')=2deg(P) therefore constant blowup in program size
  - |S|=O(|p|+|q|)=O(2k)=O(k) therefore secret is linear in the security parameter

Proof Outline – Secrecy

- Definition of β-adversary
  - β=(β1,β2,…,βm) such that Σβi=deg(P)=d)
  - An adversary that outputs the coefficient of the polynomial P (given Obf(P)) corresponding to the β-monomial x_1^(β1)x_2^(β2)…x_m^(βm)
- We show the following theorem:
  - If there exists an adversary who is either a β-adversary for any β, OR, is able to evaluate the polynomial on a random input with non-negligible advantage, then we can factor N=pq with non-negligible advantage

Proof Outline (contd)

- For the above proof we need two theorems.
- Theorem 1 (consequence of [Sha '93])
  - Consider P1 $\in$ **F** with degree d=O(log k) and P2, Q random (mod q) and (mod n) respectively
  - Set P=(P1,P2). Any adversary who can factor PQ (mod n) into P' and Q' such that PQ=P'Q' (mod n) with non-negligible advantage can factor n=pq into p and q with non-negligible advantage.

Proof Outline (contd)

- Theorem 2
  - Consider input polynomial $P \in \mathbf{F}$ and randomly chosen Q. An adversary A who does not necessarily output P, but instead is able to output P(x) (mod n) for any random input value x with non-negligible probability is powerful enough to factor n=pq with non-negligible probability
- Proofs of Theorem 1 and Theorem 2 are there in the paper (as Theorem 2, and Theorem 4 respectively)

Proof Outline (contd)

- The reduction of a factoring adversary (i.e., an adversary who solves n=pq) to β-adversary is along the lines of the proof of Theorem 1 (also there in [Sha '93])
  - This concludes the proof of security for β-adversaries
- For adversaries who evaluate the polynomial on random input, the proof of security is identical to proof of Theorem 2 (stated previously)

Elaboration on secrecy

- Why does the extension to [Sha '93] result hold?
- Why is it difficult to extract P (or Q) given PQ and P+Q?
- We show:
  - Equivalent to finding out the square root of a polynomial
  - Can reduce this to finding out the square root of a single (leading) coefficient.
  - In general (even if monic) extracting square root of leading coeffcient (mod n) is as hard as factoring
  - Hence reduces to impossibility of factoring n
  - Same hardness assumption as [Sha '93] result!

Conclusions

- New motivation and definition of hardware-assisted obfuscation
- Positive result on an interesting class of programs (with efficient practical constructions)
- Elaboration on the construction of hard-to-factor polynomials
- First provably secure obfuscation result from the difficulty of factorizing n=pq

Thank you!
Any questions?