Introduction to Obfuscation

Pooya Farshim

École Normale Supérieure

Summer School on Cryptology Crypto-CO

An Obfuscated Code

```
                                                                    char
                                                                _3141592654[3141
        ],__3141[3141];_314159[31415],_3141[31415];main(){register char*
     _3_141,*_3_1415, *_3__1415; register int _314,_31415,__31415,*_31,
   _3_14159,__3_1415;*_3141592654=__31415=2,_3141592654[0][_3141592654
  -1]=1[__3141]=5;__3_1415=1;do{_3_14159=_314=0,__31415++;for( _31415
 =0;_31415<(3,14-4)*__31415;_31415++)_31415[_3141]=_314159[_31415]= -
1;_3141[*_314159=_3_14159]=_314;_3_141=_3141592654+__3_1415;_3_1415=
__3_1415    +__3141;for              (_31415 = 3141-
          __3_1415  ;                _31415;_31415--
          ,_3_141 ++,                _3_1415++){_314
          +=_314<<2 ;                _314<<=1;_314+=
         *_3_1415;_31                 =_314159+_314;
         if(!(*_31+1)                 )* _31 =_314 /
         __31415,_314                 [_3141]=_314 %
         __31415 ;* (                 _3__1415=_3_141
        )+= *_3_1415                   = *_31;while(*
        _3__1415 >=                    31415/3141 ) *
        _3__1415+= -                   10,(*--_3__1415
       )++;_314=_314                   [_3141]; if ( !
       _3_14159 && *                   _3_1415)_3_14159
       =1,__3_1415 =                   3141-_31415;}if(
       _314+(__31415                    >>1)>=__31415 )
       while ( ++ *                     _3_141==3141/314
      )*_3_141--=0                      ;}while(_3_14159
      ) ; { char *                      __3_14= "3.1415";
      write((3,1),                      (--*__3_14,__3_14
      ),(_3_14159                        ++,++_3_14159))+
     3.1415926; }                        for ( _31415 = 1;
    _31415<3141-                         1;_31415++)write(
   31415% 314-(                          3,14),_3141592654[
  _31415    ] +                         "0123456789","314"
  [ 3]+1)-_314;                         puts((*_3141592654=0
,_3141592654))                           ;_314= *"3.141592";}
```

What's Obfuscation?

Obfuscation is the task of **making programs unintelligible.**

What's Obfuscation?

Obfuscation is the task of **making programs unintelligible.**

$$\text{Program } P \quad \overset{\text{Obf}}{\rightsquigarrow} \quad \text{Program } \overline{P}$$

Such that:

- Programs P and $\overline{P}$ have the same functionality.
- $\overline{P}$ is unintelligible.

What's Obfuscation?

Obfuscation is the task of **making programs unintelligible.**

$$\text{Program } P \quad \overset{\text{Obf}}{\rightsquigarrow} \quad \text{Program } \overline{P}$$

Such that:

- Programs P and $\overline{P}$ have the same functionality.
- $\overline{P}$ is unintelligible.

In a sense, we want to "encrypt programs" (rather than messages).

Why Look at Obfuscators?

With a general-purpose obfuscator:

- None can we run arbitrary programs
- while **hiding secrets** within those programs.
- And have guarantees that the program code does not leak any unintentional information.

Potential Applications

- **Software protection**: Publish software resilient to reverse-engineering:

  Publish $\mathrm{Obf}(C)$ instead of C.

Potential Applications

- **Software protection**: Publish software resilient to reverse-engineering:

$$\text{Publish } \mathsf{Obf}(C) \text{ instead of } C.$$

- **Hide keys:** Give a symmetric encryption (SE) convert it to a PKE:

$$\text{Distribute } pk := \mathsf{Obf}(\mathsf{Enc}(k, \cdot))$$

A circuit that allows encryption under k without revealing k.

Potential Applications

- **Software protection**: Publish software resilient to reverse-engineering:

$$\text{Publish } \mathsf{Obf}(C) \text{ instead of } C.$$

- **Hide keys:** Give a symmetric encryption (SE) convert it to a PKE:

$$\text{Distribute } pk := \mathsf{Obf}(\mathsf{Enc}(k, \cdot))$$

  A circuit that allows encryption under k without revealing k.

- **Hash Functions**: Public random functions:

$$\mathsf{H}(\cdot) := \mathsf{Obf}(\mathsf{AES}(k, \cdot))$$

Potential Applications

A **homomorphic encryption** scheme allows

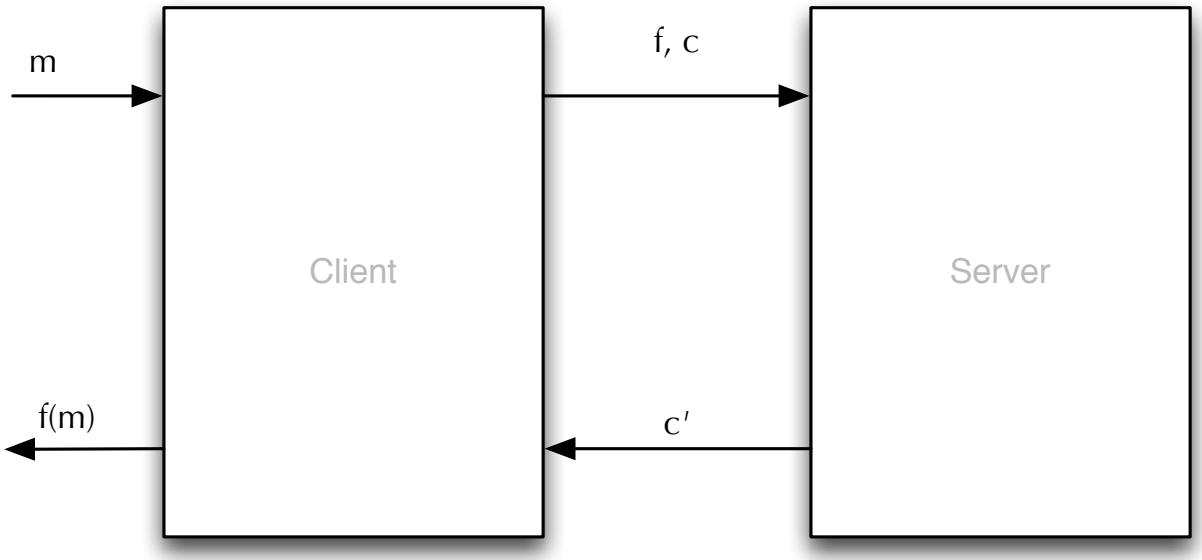
Computing $\mathsf{Enc}(k, f(m))$ from $\mathsf{Enc}(k, m)$ **without** k.

Potential Applications

A **homomorphic encryption** scheme allows

Computing $\mathsf{Enc}(k, f(m))$ from $\mathsf{Enc}(k, m)$ **without** k.

We can use this to **privately outsource computation**:

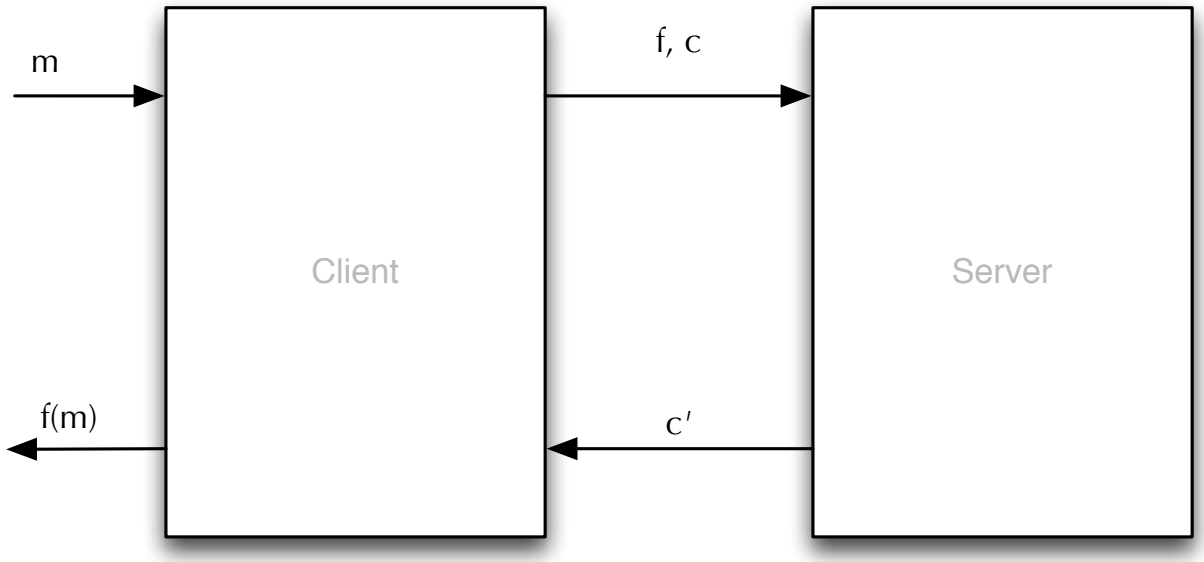


Potential Applications

A **homomorphic encryption** scheme allows

$$\text{Computing } \mathsf{Enc}(k, f(m)) \text{ from } \mathsf{Enc}(k, m) \textbf{ without } k.$$

We can use this to **privately outsource computation**:



Solution: Obfuscate "decrypt, apply f, re-encrypt":

$$\text{Publish } \mathsf{Eval}(f, c) := \mathsf{Obf}\Big(\mathsf{Enc}\big(k, f\big(\mathsf{Dec}(k, c)\big)\big) \Big)$$

Functional Encryption

Traditional Public-Key Encryption is **all-or-nothing**:

> You either know sk and can recover m fully from $\mathsf{Enc}(pk, m)$
> or you don't know it, and you don't get any information on m.

Functional Encryption

Traditional Public-Key Encryption is **all-or-nothing**:

> You either know sk and can recover m fully from $\mathsf{Enc}(pk, m)$
> or you don't know it, and you don't get any information on m.

Functional Encryption extends PKEs to a **fine-grained** setting:

> For each function f an associated secret key sk_f allows
> recovering $f(m)$ from $\mathsf{Enc}(pk, m)$ and nothing more.

Solution: Obfuscate "Decrypt then apply f":

$$sk_f := \mathsf{Obf}\big(\ f(\mathsf{Dec}(sk, \cdot))\ \big)$$

Functional Encryption

Traditional Public-Key Encryption is **all-or-nothing**:

> You either know sk and can recover m fully from $\mathsf{Enc}(pk, m)$
> or you don't know it, and you don't get any information on m.

Functional Encryption extends PKEs to a **fine-grained** setting:

> For each function f an associated secret key sk_f allows
> recovering $f(m)$ from $\mathsf{Enc}(pk, m)$ and nothing more.

Solution: Obfuscate "Decrypt then apply f":

$$sk_f := \mathsf{Obf}\big(\ f(\mathsf{Dec}(sk, \cdot))\ \big)$$

(**Note:** Difference with hom. enc. is that output is in the clear.)

A Simple Application

Consider the program:

$$C[pwd](x) := \begin{cases} \text{Success} & \text{if } x = pwd; \\ \text{Access denied} & \text{otherwise.} \end{cases}$$

Useful for log-in applications.

A Simple Application

Consider the program:

$$C[pwd](x) := \begin{cases} \text{Success} & \text{if } x = pwd; \\ \text{Access denied} & \text{otherwise.} \end{cases}$$

Useful for log-in applications.

Obfuscation ensures pwd remains hidden given a code for $C[pwd]$.

Potential Applications

Many other applications:

Potential Applications

Many other applications:

- Proxy Re-Encryption

$$\text{Convert} \qquad \text{Enc}(pk_A, m) \qquad \text{to} \qquad \text{Enc}(pk_B, m)$$

Potential Applications

Many other applications:

- Proxy Re-Encryption

$$\text{Convert} \qquad \text{Enc}(pk_A, m) \qquad \text{to} \qquad \text{Enc}(pk_B, m)$$

- Watermarking software
- ...

Obfuscation is an extremely powerful tool:

A "crypto-complete" primitive, a "central hub" for crypto, ...

Syntax and Correctness

Syntax: An obfuscator is an efficient randomized algorithm

- $\mathsf{Obf}(\lambda, \mathsf{C})$ that on input a circuit C outputs another circuit $\overline{\mathsf{C}}$.

Syntax and Correctness

Syntax: An obfuscator is an efficient randomized algorithm

- $\mathsf{Obf}(\lambda, \mathsf{C})$ that on input a circuit C outputs another circuit $\overline{\mathsf{C}}$.

Correctness: For all $\lambda \in \mathbb{N}$, all circuits $\mathsf{C} \in \mathcal{C}$, and any $\overline{\mathsf{C}} \leftarrow\!\!\$\ \mathsf{Obf}(1^\lambda, \mathsf{C})$:

$$(\forall x) \ : \ \overline{\mathsf{C}}(x) = \mathsf{C}(x)$$

Ideally, we want $\mathcal{C}$ to be the class of **all** circuits.

Syntax and Correctness

Syntax: An obfuscator is an efficient randomized algorithm

- $\mathsf{Obf}(\lambda, \mathsf{C})$ that on input a circuit C outputs another circuit $\overline{\mathsf{C}}$.

Correctness: For all $\lambda \in \mathbb{N}$, all circuits $\mathsf{C} \in \mathcal{C}$, and any $\overline{\mathsf{C}} \leftarrow\!\!\$\ \mathsf{Obf}(1^\lambda, \mathsf{C})$:

$$(\forall x) \ : \ \overline{\mathsf{C}}(x) = \mathsf{C}(x)$$

Ideally, we want $\mathcal{C}$ to be the class of **all** circuits.

Efficiency: $\mathsf{Obf}(C)$ compiles C to a program $\overline{\mathsf{C}}$ that is polynomially slower than C.

Security

What should we expect from a good obfuscator?

Security

What should we expect from a good obfuscator?

An obfuscated code is essentially a **virtual black box**:
All you can do with an obfuscated code is to run it on some inputs.

Security

What should we expect from a good obfuscator?

An obfuscated code is essentially a **virtual black box**:
All you can do with an obfuscated code is to run it on some inputs.

A first try: For all efficient $\mathcal{A}$

$$\mathcal{A}(\mathsf{Obf}(C)) \quad \overset{c}{\approx} \quad \mathcal{A}^{C} \ .$$

$\mathcal{A}$'s I/O fixed: we need ensure this formally makes sense.

Security

What should we expect from a good obfuscator?

An obfuscated code is essentially a **virtual black box**:
All you can do with an obfuscated code is to run it on some inputs.

A first try: For all efficient $\mathcal{A}$

$$\mathcal{A}(\mathsf{Obf}(C)) \quad \stackrel{c}{\approx} \quad \mathcal{A}^C \ .$$

$\mathcal{A}$'s I/O fixed: we need ensure this formally makes sense.

VBB Security: For all efficient $\mathcal{A}$ there is an efficient **simulator** $\mathcal{S}$ such that for all circuits C:

$$\mathcal{A}(\mathsf{Obf}(C)) \quad \stackrel{c}{\approx} \quad \mathcal{S}^C \ .$$

[BGI+01]

On the (Im)possibility of Obfuscating Programs*

Boaz Barak† Oded Goldreich‡ Russell Impagliazzo§ Steven Rudich¶
Amit Sahai‖ Salil Vadhan** Ke Yang††

July 29, 2010

Abstract

Informally, an *obfuscator* $\mathcal{O}$ is an (efficient, probabilistic) "compiler" that takes as input a program (or circuit) P and produces a new program $\mathcal{O}(P)$ that has the same functionality as P yet is "unintelligible" in some sense. Obfuscators, if they exist, would have a wide variety of cryptographic and complexity-theoretic applications, ranging from software protection to homomorphic encryption to complexity-theoretic analogues of Rice's theorem. Most of these applications are based on an interpretation of the "unintelligibility" condition in obfuscation as meaning that $\mathcal{O}(P)$ is a "virtual black box," in the sense that anything one can efficiently compute given $\mathcal{O}(P)$, one could also efficiently compute given oracle access to P.

In this work, we initiate a theoretical investigation of obfuscation. Our main result is that, even under very weak formalizations of the above intuition, obfuscation is impossible. We prove this by constructing a family of efficient programs $\mathcal{P}$ that are *unobfuscatable* in the sense that (a) given *any* efficient program P' that computes the same function as a program $P \in \mathcal{P}$, the "source code" P can be efficiently reconstructed, yet (b) given *oracle access* to a (randomly selected) program $P \in \mathcal{P}$, no efficient algorithm can reconstruct P (or even distinguish a certain bit in the code from random) except with negligible probability.

We extend our impossibility result in a number of ways, including even obfuscators that

Feasibility

Let's start with

$$C[x^*](x) := \begin{cases} 1 & \text{if } x = x^*; \\ 0 & \text{otherwise.} \end{cases}$$

Feasibility

Let's start with

$$C[x^*](x) := \begin{cases} 1 & \text{if } x = x^*; \\ 0 & \text{otherwise.} \end{cases}$$

Let f be a one-way permutation. Define $\mathsf{Obf}(C[x^*](x))$ as:

- Let $y^* := \pi(x^*)$
- Return the program:

$$\overline{C}[y^*](x) := \begin{cases} 1 & \text{if } \pi(x) = y^*; \\ 0 & \text{otherwise.} \end{cases}$$

Feasibility

Let's start with

$$C[x^*](x) := \begin{cases} 1 & \text{if } x = x^*; \\ 0 & \text{otherwise.} \end{cases}$$

Let f be a one-way permutation. Define $\mathsf{Obf}(C[x^*](x))$ as:

- Let $y^* := \pi(x^*)$
- Return the program:

$$\overline{C}[y^*](x) := \begin{cases} 1 & \text{if } \pi(x) = y^*; \\ 0 & \text{otherwise.} \end{cases}$$

Correctness: $\overline{C}$ preserves functionality as π is a permutation.
Security: Intuition: Think of π as being ideal.

Bad News

$\mathcal{A}$ can run the obfuscated code on itself, but $\mathcal{S}$ cannot.

Can be used to rule out the existence of general-purpose VBB obfuscation.

Bad News

$\mathcal{A}$ can run the obfuscated code on itself, but $\mathcal{S}$ cannot.

Can be used to rule out the existence of general-purpose VBB obfuscation.

Using other techniques one can show:

VBB obfuscation is **impossible** for wide classes of circuits,
including many cryptographically interesting ones.

Bad News

$\mathcal{A}$ can run the obfuscated code on itself, but $\mathcal{S}$ cannot.

Can be used to rule out the existence of general-purpose VBB obfuscation.

Using other techniques one can show:

VBB obfuscation is **impossible** for wide classes of circuits,
including many cryptographically interesting ones.

This leads to the question:

Are there weaker (or incomparable) notions of security
that are both feasible and useful?

Breakthrough Result of 2013

Candidate Indistinguishability Obfuscation and Functional Encryption for all circuits

Sanjam Garg
UCLA
sanjamg@cs.ucla.edu

Craig Gentry
IBM Research
craigbgentry@gmail.com

Shai Halevi
IBM Research
shaih@alum.mit.edu

Mariana Raykova
IBM Research
mariana@cs.columbia.edu

Amit Sahai
UCLA
sahai@cs.ucla.edu

Brent Waters
University of Texas at Austin
bwaters@cs.utexas.edu

July 21, 2013

Abstract

In this work, we study *indistinguishability obfuscation* and *functional encryption* for general circuits:

Indistinguishability obfuscation requires that given any two equivalent circuits C_0 and C_1 of similar size, the obfuscations of C_0 and C_1 should be computationally indistinguishable.

In functional encryption, ciphertexts encrypt inputs x and keys are issued for circuits C. Using the key SK_C to decrypt a ciphertext $\mathrm{CT}_x = \mathsf{Enc}(x)$, yields the value $C(x)$ but does not reveal anything else about x. Furthermore, no collusion of secret key holders should be able to learn anything more than the union of what they can each learn individually.

We give constructions for indistinguishability obfuscation and functional encryption that supports all polynomial-size circuits. We accomplish this goal in three steps:

- We describe a candidate construction for indistinguishability obfuscation for $\mathbf{NC}^1$ circuits. The security of this construction is based on a new algebraic hardness assumption. The candidate and assumption use a simplified variant of multilinear maps, which we call *Multilinear Jigsaw Puzzles*.
- We show how to use indistinguishability obfuscation for $\mathbf{NC}^1$ together with Fully Homomorphic Encryption (with decryption in $\mathbf{NC}^1$) to achieve indistinguishability obfuscation for all circuits.
- Finally, we show how to use indistinguishability obfuscation for circuits, public-key encryption, and non-interactive zero knowledge to achieve functional encryption for all circuits. The functional encryption scheme we construct also enjoys succinct ciphertexts, which enables several other applications.

Indistinguishability Obfuscation (iO)

Call two circuits C_0 and C_1 **functionally equivalent** if

$$(\forall x) \ : \quad C_0(x) = C_1(x) \ .$$

Indistinguishability Obfuscation (iO)

Call two circuits C_0 and C_1 **functionally equivalent** if

$$(\forall x) \ : \quad C_0(x) = C_1(x) \ .$$

IND Security: No adversary can distinguish obfuscations of such circuits:

$$\mathsf{Obf}(C_0) \quad \overset{c}{\approx} \quad \mathsf{Obf}(C_1) \ .$$

Indistinguishability Obfuscation (iO)

Call two circuits C_0 and C_1 **functionally equivalent** if

$$(\forall x) \ : \quad C_0(x) = C_1(x) \ .$$

IND Security: No adversary can distinguish obfuscations of such circuits:

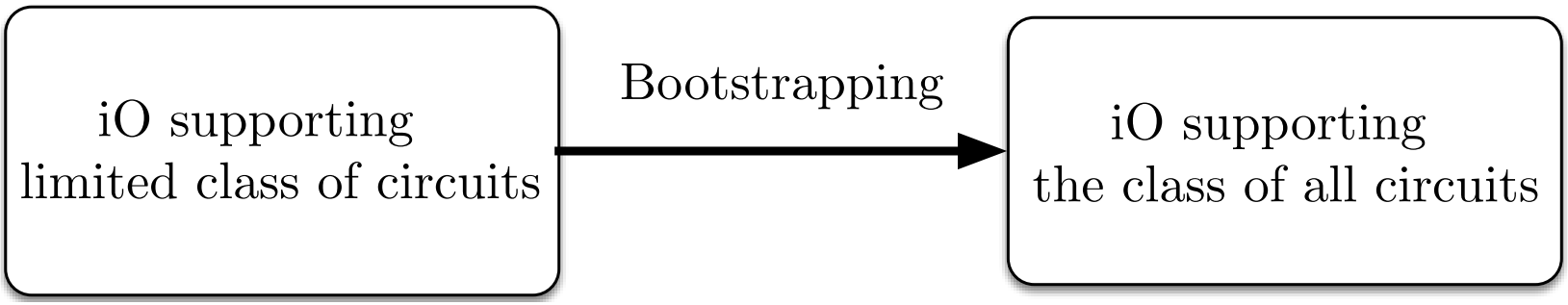
$$\mathsf{Obf}(C_0) \quad \overset{c}{\approx} \quad \mathsf{Obf}(C_1) \ .$$

For instance, the obfuscations of

$$C_0(x, y) := (x + y)^2 \quad \text{and} \quad C_1(x, y) := (x^2 + y^2 + 2xy) \ .$$

should look indistinguishable.

iO Construction



iO for $\mathbf{C}_{\text{small}}$

Recall bilinear maps (aka. pairings):

$$e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$$

$$e(g^a, g^b) = e(g, g)^{ab}$$

iO for $\mathbf{C}_{\text{small}}$

Recall bilinear maps (aka. pairings):

$$e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$$
$$e(g^a, g^b) = e(g, g)^{ab}$$

For iO we need k-linear maps:

$$e : \mathbb{G} \times \cdots \times \mathbb{G} \longrightarrow \mathbb{G}_T$$
$$e(g^{a_1}, \ldots, g^{a_k}) = e(g, \ldots, g)^{a_1 \cdots a_k}$$

- Allow computing polynomials of degree $\leq k$ in the exponent.
- Checking if the output was zero or one.
- Multi-linear map security $\implies$ iO security

Fully Homomorphic Encryption

1. $\mathsf{Gen}(\lambda)$ outputs a key pair (sk, pk)
2. $\mathsf{Enc}(pk, m)$ outputs a ciphertext c.
3. $\mathsf{Eval}(pk, f, c)$ outputs an evaluated ciphertext c.
4. $\mathsf{Dec}(sk, c)$ outputs a message m.

Fully Homomorphic Encryption

1. $\mathsf{Gen}(\lambda)$ outputs a key pair (sk, pk)
2. $\mathsf{Enc}(pk, m)$ outputs a ciphertext c.
3. $\mathsf{Eval}(pk, f, c)$ outputs an evaluated ciphertext c.
4. $\mathsf{Dec}(sk, c)$ outputs a message m.

Correctness: If c encrypt m, then

$$\mathsf{Dec}(sk, \mathsf{Eval}(pk, f, c)) = f(m) \ .$$

Fully Homomorphic Encryption

1. $\mathsf{Gen}(\lambda)$ outputs a key pair (sk, pk)
2. $\mathsf{Enc}(pk, m)$ outputs a ciphertext c.
3. $\mathsf{Eval}(pk, f, c)$ outputs an evaluated ciphertext c.
4. $\mathsf{Dec}(sk, c)$ outputs a message m.

Correctness: If c encrypt m, then

$$\mathsf{Dec}(sk, \mathsf{Eval}(pk, f, c)) = f(m) \ .$$

Security: Identical to IND-CPA security.

$$(pk, \mathsf{Enc}(pk, m_0)) \overset{c}{\approx} (pk, \mathsf{Enc}(pk, m_1))$$

The Bootstrapping Step

Intuition:

$$\mathsf{Obf}(C) := \big(\mathsf{Enc}(pk, C), \mathsf{iO}(\mathsf{Dec}(sk, \cdot)\big).$$

The Bootstrapping Step

Intuition:

$$\mathsf{Obf}(C) := \big(\mathsf{Enc}(pk, C), \mathsf{iO}(\mathsf{Dec}(sk, \cdot)\big).$$

How can we run $\mathsf{Obf}(C)$ on an input x?

The Bootstrapping Step

Intuition:

$$\mathsf{Obf}(C) := \big(\mathsf{Enc}(pk, C), \mathsf{iO}(\mathsf{Dec}(sk, \cdot)\big).$$

How can we run $\mathsf{Obf}(C)$ on an input x? Let $\mathrm{Universal}$ be a universal Turing machine:

$$\mathrm{Universal}(C, x) = C(x)$$

The Bootstrapping Step

Intuition:

$$\mathsf{Obf}(C) := \big(\mathsf{Enc}(pk, C), \mathsf{iO}(\mathsf{Dec}(sk, \cdot)\big).$$

How can we run $\mathsf{Obf}(C)$ on an input x? Let $\mathrm{Universal}$ be a universal Turing machine:

$$\mathrm{Universal}(C, x) = C(x)$$

Now proceed as follows:

- Run $\mathsf{Eval}(pk, \mathrm{Universal}(\cdot, x), \mathsf{Enc}(pk, C))$ to get c'. Ciphertext c' now encrypts $C(x)$.
- Run $\mathsf{iO}(\mathsf{Dec}(sk, \cdot))$ on c' to decrypt it and recover $C(x)$.

An Application of iO

The Symmetric-to-Asymmetric Conversion

SE-to-PKE Conversion

Recall one consequence of VBB obfuscation was:

We can convert a symmetric encryption to a public-key one.

$$pk := \mathsf{Obf}(\mathsf{Enc}(k, \cdot))$$

SE-to-PKE Conversion

Recall one consequence of VBB obfuscation was:

We can convert a symmetric encryption to a public-key one.

$$pk := \mathsf{Obf}(\mathsf{Enc}(k, \cdot))$$

Natural question: Can this be done with iO?

SE-to-PKE Conversion

Recall one consequence of VBB obfuscation was:

We can convert a symmetric encryption to a public-key one.

$$pk := \mathsf{Obf}(\mathsf{Enc}(k, \cdot))$$

Natural question: Can this be done with iO?

- Yes, but we now need a special type of Enc.

Tools

Some more foundational primitives.

Pseudo-random Generators (PRGs)

Syntax: A PRG is deterministic length-increasing function:

$$\mathsf{PRG} : \{0,1\}^n \longrightarrow \{0,1\}^{n+s} \ .$$

Pseudo-random Generators (PRGs)

Syntax: A PRG is deterministic length-increasing function:

$$\mathsf{PRG} : \{0,1\}^n \longrightarrow \{0,1\}^{n+s} \ .$$

Security: For a random seed s and a truly random $y \in \{0,1\}^{n+s}$

$$\mathsf{PRG}(s) \quad \overset{c}{\approx} \quad y$$

Pseudo-random Generators (PRGs)

Syntax: A PRG is deterministic length-increasing function:

$$\mathsf{PRG} : \{0,1\}^n \longrightarrow \{0,1\}^{n+s} \ .$$

Security: For a random seed s and a truly random $y \in \{0,1\}^{n+s}$

$$\mathsf{PRG}(s) \quad \overset{c}{\approx} \quad y$$

Observations:

Pseudo-random Generators (PRGs)

Syntax: A PRG is deterministic length-increasing function:

$$\text{PRG} : \{0,1\}^n \longrightarrow \{0,1\}^{n+s} \ .$$

Security: For a random seed s and a truly random $y \in \{0,1\}^{n+s}$

$$\text{PRG}(s) \quad \overset{c}{\approx} \quad y$$

Observations:

- A PRG **stretches** n bits of randomness to $(n+s)$ bits.

Pseudo-random Generators (PRGs)

Syntax: A PRG is deterministic length-increasing function:

$$\mathsf{PRG} : \{0,1\}^n \longrightarrow \{0,1\}^{n+s} \ .$$

Security: For a random seed s and a truly random $y \in \{0,1\}^{n+s}$

$$\mathsf{PRG}(s) \quad \overset{c}{\approx} \quad y$$

Observations:

- A PRG **stretches** n bits of randomness to $(n+s)$ bits.
- By **iterating** we can build a PRG with any stretch.

Pseudo-random Generators (PRGs)

Syntax: A PRG is deterministic length-increasing function:

$$\mathsf{PRG} : \{0,1\}^n \longrightarrow \{0,1\}^{n+s} \ .$$

Security: For a random seed s and a truly random $y \in \{0,1\}^{n+s}$

$$\mathsf{PRG}(s) \quad \overset{c}{\approx} \quad y$$

Observations:

- A PRG **stretches** n bits of randomness to $(n+s)$ bits.
- By **iterating** we can build a PRG with any stretch.
- Prove PRG $\implies$ OWF.

Pseudo-random Generators (PRGs)

Syntax: A PRG is deterministic length-increasing function:

$$\mathsf{PRG} : \{0,1\}^n \longrightarrow \{0,1\}^{n+s} \ .$$

Security: For a random seed s and a truly random $y \in \{0,1\}^{n+s}$

$$\mathsf{PRG}(s) \quad \overset{c}{\approx} \quad y$$

Observations:

- A PRG **stretches** n bits of randomness to $(n+s)$ bits.
- By **iterating** we can build a PRG with any stretch.
- Prove PRG $\implies$ OWF.
- The other direction, OWF $\implies$ PRG, is much harder. One of the main classical results in crypto.

Pseudo-random Functions (PRFs)

Syntax: A PRF is deterministic keyed function

$$\mathsf{PRF} : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^m \ .$$

Pseudo-random Functions (PRFs)

Syntax: A PRF is deterministic keyed function

$$\mathsf{PRF} : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^m \ .$$

Security: For a random key k, the outputs of PRF look indistinguishable from random: For any efficient adversary $\mathcal{A}$:

$$\mathcal{A}^{\mathsf{PRF}(k,\cdot)} \quad \overset{c}{\approx} \quad \mathcal{A}^{\mathsf{RO}(\cdot)}$$

Here RO is a **Random Oracle** (aka. random function):

- On each input x it returns a truly random value $y := \mathsf{RO}(x)$
- On repeat inputs it's consistent and repeats outputs.

Pseudo-random Functions (PRFs)

Syntax: A PRF is deterministic keyed function

$$\mathrm{PRF} : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^m \ .$$

Security: For a random key k, the outputs of PRF look indistinguishable from random: For any efficient adversary $\mathcal{A}$:

$$\mathcal{A}^{\mathrm{PRF}(k,\cdot)} \quad \stackrel{c}{\approx} \quad \mathcal{A}^{\mathrm{RO}(\cdot)}$$

Here RO is a **Random Oracle** (aka. random function):

- On each input x it returns a truly random value $y := \mathrm{RO}(x)$
- On repeat inputs it's consistent and repeats outputs.

Prove: PRF $\implies$ PRG. (Use PRF a few times.)
Fact: The GGM construction shows: PRG $\implies$ PRF.

Summary

PRG : a function that stretch random strings form n bits to $m > n$ bits.

PRF : a function that behaves like a random functions as long as its key remains hidden.

The Conversion

The Symmetric-to-Asymmetric Conversion

A Basic Symmetric Encryption

Consider the SE scheme:

- Gen(λ): Return a random key $k \in \{0,1\}^{\lambda}$

A Basic Symmetric Encryption

Consider the SE scheme:

- $\mathsf{Gen}(\lambda)$: Return a random key $k \in \{0,1\}^{\lambda}$
- $\mathsf{Enc}(k, m; r)$: For a random $r \in \{0,1\}^{2\lambda}$ return

$$(c_1, c_2) := (r, \mathsf{PRF}(k, r) \oplus m)$$

A Basic Symmetric Encryption

Consider the SE scheme:

- $\mathsf{Gen}(\lambda)$: Return a random key $k \in \{0,1\}^\lambda$
- $\mathsf{Enc}(k, m; r)$: For a random $r \in \{0,1\}^{2\lambda}$ return

$$(c_1, c_2) := (r, \mathsf{PRF}(k, r) \oplus m)$$

- $\mathsf{Dec}(k, (c_1, c_2))$: Return

$$m := \mathsf{PRF}(k, c_1) \oplus c_2$$

A Basic Symmetric Encryption

Consider the SE scheme:

- $\mathsf{Gen}(\lambda)$: Return a random key $k \in \{0,1\}^\lambda$
- $\mathsf{Enc}(k, m; r)$: For a random $r \in \{0,1\}^{2\lambda}$ return

$$(c_1, c_2) := (r, \mathsf{PRF}(k, r) \oplus m)$$

- $\mathsf{Dec}(k, (c_1, c_2))$: Return

$$m := \mathsf{PRF}(k, c_1) \oplus c_2$$

Prove: This scheme is IND-CPA secure if the PRF is secure.
Intuition: XORing with PRF acts as a one-time pad.

Two Modifications

$$\mathsf{Enc}(k, m; r) := (r, \mathsf{PRF}(k, r) \oplus m)$$

Two modifications:

Two Modifications

$$\mathsf{Enc}(k, m; r) := (r, \mathsf{PRF}(k, r) \oplus m)$$

Two modifications:

- Use a PRG: $\{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$ to generate r:

$$\mathsf{Enc}(k, m; s) := (r := \mathsf{PRG}(s), \mathsf{PRF}(k, r) \oplus m)$$

Two Modifications

$$\mathsf{Enc}(k, m; r) := (r, \mathsf{PRF}(k, r) \oplus m)$$

Two modifications:

- Use a PRG: $\{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$ to generate r:

$$\mathsf{Enc}(k, m; s) := (r := \mathsf{PRG}(s), \mathsf{PRF}(k, r) \oplus m)$$

- We need to use a special type of PRF in place of a standard PRF.

Exercise: Prove this scheme is still IND-CPA secure.
Intuition: PRG is secure, so randomness r looks random. The rest is as before.

The SE-to-PKE Conversion

Given the scheme $\mathsf{SE} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ above:

The SE-to-PKE Conversion

Given the scheme $\mathsf{SE} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ above:

- $\underline{\mathsf{PKE.Gen}(\lambda)}$: Run $\mathsf{Gen}(\lambda)$ to get k. Set

$$sk := k \qquad pk := \mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$$

The SE-to-PKE Conversion

Given the scheme $\mathsf{SE} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ above:

- $\underline{\mathsf{PKE.Gen}(\lambda)}$: Run $\mathsf{Gen}(\lambda)$ to get k. Set

$$sk := k \qquad pk := \mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$$

- $\underline{\mathsf{PKE.Enc}(pk, m; s)}$: Run pk on m, s. Note the output is

$$(c_1, c_2) := (\mathsf{PRG}(s), \mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m))$$

The SE-to-PKE Conversion

Given the scheme $\mathsf{SE} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ above:

- $\underline{\mathsf{PKE.Gen}(\lambda)}$: Run $\mathsf{Gen}(\lambda)$ to get k. Set

$$sk := k \qquad pk := \mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$$

- $\underline{\mathsf{PKE.Enc}(pk, m; s)}$: Run pk on m, s. Note the output is

$$(c_1, c_2) := (\mathsf{PRG}(s), \mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m))$$

- $\underline{\mathsf{PKE.Dec}(sk, (c_1, c_2))}$: Run $\mathsf{Dec}(k, (c_1, c_2))$

The SE-to-PKE Conversion

Given the scheme $\mathsf{SE} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ above:

- $\mathsf{PKE.Gen}(\lambda)$: Run $\mathsf{Gen}(\lambda)$ to get k. Set

$$sk := k \qquad pk := \mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$$

- $\mathsf{PKE.Enc}(pk, m; s)$: Run pk on m, s. Note the output is

$$(c_1, c_2) := (\mathsf{PRG}(s), \mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m))$$

- $\mathsf{PKE.Dec}(sk, (c_1, c_2))$: Run $\mathsf{Dec}(k, (c_1, c_2))$

Correctness: Follows form the correctness of SE and the correctness of iO.
Security: Takes 2/3 more slides.

Security Proof

Theorem

For any two messages m_0 and m_1:

$$(pk, \mathsf{Enc}(pk, m_0)) \overset{c}{\approx} (pk, \mathsf{Enc}(pk, m_1))$$

$$(\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_0)}_{c_2}) \overset{c}{\approx} (\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_1)}_{c_2})$$

Proof.

Gm	pk	c_1	c_2	**Why?**
0	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$r^* := \mathsf{PRG}(s)$	$m_b \oplus \mathsf{PRF}(k, r^*)$	
1	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$\boxed{r^* \leftarrow \{0,1\}^{2\lambda}}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	Security of PRG
2	$\boxed{\mathsf{iO}(\mathsf{Enc}(k_{r^*}, \cdot; \cdot))}$	$r^* \leftarrow \{0,1\}^{2\lambda}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	Security of iO
3	$\mathsf{iO}(\mathsf{Enc}(k_{r^*}, \cdot; \cdot))$	$r^* \leftarrow \{0,1\}^{2\lambda}$	$\boxed{m_b \oplus \text{ Random}}$	Security of PuncPRF Adv $= 0$

Security Proof

Theorem

For any two messages m_0 and m_1:

$$(pk, \mathsf{Enc}(pk, m_0)) \overset{c}{\approx} (pk, \mathsf{Enc}(pk, m_1))$$

$$(\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_0)}_{c_2}) \overset{c}{\approx} (\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_1)}_{c_2})$$

Security Proof

Theorem

For any two messages m_0 and m_1:

$$(pk, \mathsf{Enc}(pk, m_0)) \overset{c}{\approx} (pk, \mathsf{Enc}(pk, m_1))$$

$$(\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_0)}_{c_2}) \overset{c}{\approx} (\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_1)}_{c_2})$$

Proof.

Gm	pk	c_1	c_2	**Why?**
0	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$r^* := \mathsf{PRG}(s)$	$m_b \oplus \mathsf{PRF}(k, r^*)$	

Security Proof

Theorem

For any two messages m_0 and m_1:

$$(pk, \mathsf{Enc}(pk, m_0)) \overset{c}{\approx} (pk, \mathsf{Enc}(pk, m_1))$$

$$(\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_0)}_{c_2}) \overset{c}{\approx} (\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_1)}_{c_2})$$

Proof.

Gm	pk	c_1	c_2	**Why?**
0	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$r^* := \mathsf{PRG}(s)$	$m_b \oplus \mathsf{PRF}(k, r^*)$	
1	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$\boxed{r^* \leftarrow \{0,1\}^{2\lambda}}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	

Security Proof

Theorem

For any two messages m_0 and m_1:

$$(pk, \mathsf{Enc}(pk, m_0)) \overset{c}{\approx} (pk, \mathsf{Enc}(pk, m_1))$$

$$(\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_0)}_{c_2}) \overset{c}{\approx} (\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_1)}_{c_2})$$

Proof.

Gm	pk	c_1	c_2	**Why?**
0	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$r^* := \mathsf{PRG}(s)$	$m_b \oplus \mathsf{PRF}(k, r^*)$	
1	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$\boxed{r^* \leftarrow \{0,1\}^{2\lambda}}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	Security of PRG

Security Proof

Theorem

For any two messages m_0 and m_1:

$$(pk, \mathsf{Enc}(pk, m_0)) \overset{c}{\approx} (pk, \mathsf{Enc}(pk, m_1))$$

$$(\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_0)}_{c_2}) \overset{c}{\approx} (\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_1)}_{c_2})$$

Proof.

Gm	pk	c_1	c_2	**Why?**
0	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$r^* := \mathsf{PRG}(s)$	$m_b \oplus \mathsf{PRF}(k, r^*)$	
1	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$r^* \leftarrow \{0,1\}^{2\lambda}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	Security of PRG
2	$\mathsf{iO}(\mathsf{Enc}(k_{r^*}, \cdot; \cdot))$	$r^* \leftarrow \{0,1\}^{2\lambda}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	Security of iO

Security Proof

Theorem

For any two messages m_0 and m_1:

$$(pk, \mathsf{Enc}(pk, m_0)) \overset{c}{\approx} (pk, \mathsf{Enc}(pk, m_1))$$

$$(\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_0)}_{c_2}) \overset{c}{\approx} (\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_1)}_{c_2})$$

Proof.

Gm	pk	c_1	c_2	**Why?**
0	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$r^* := \mathsf{PRG}(s)$	$m_b \oplus \mathsf{PRF}(k, r^*)$	
1	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$\boxed{r^* \leftarrow \{0,1\}^{2\lambda}}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	Security of PRG
2	$\boxed{\mathsf{iO}(\mathsf{Enc}(k_{r^*}, \cdot; \cdot))}$	$r^* \leftarrow \{0,1\}^{2\lambda}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	Security of iO
3	$\mathsf{iO}(\mathsf{Enc}(k_{r^*}, \cdot; \cdot))$	$r^* \leftarrow \{0,1\}^{2\lambda}$	$\boxed{m_b \oplus \ \mathsf{Random}}$	Security of PuncPRF

Security Proof

Theorem

For any two messages m_0 and m_1:

$$(pk, \mathsf{Enc}(pk, m_0)) \overset{c}{\approx} (pk, \mathsf{Enc}(pk, m_1))$$

$$(\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_0)}_{c_2}) \overset{c}{\approx} (\underbrace{\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))}_{pk}, \underbrace{\mathsf{PRG}(s)}_{c_1}, \underbrace{\mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m_1)}_{c_2})$$

Proof.

Gm	pk	c_1	c_2	**Why?**
0	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$r^* := \mathsf{PRG}(s)$	$m_b \oplus \mathsf{PRF}(k, r^*)$	
1	$\mathsf{iO}(\mathsf{Enc}(k, \cdot; \cdot))$	$\boxed{r^* \leftarrow \{0,1\}^{2\lambda}}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	Security of PRG
2	$\boxed{\mathsf{iO}(\mathsf{Enc}(k_{r^*}, \cdot; \cdot))}$	$r^* \leftarrow \{0,1\}^{2\lambda}$	$m_b \oplus \mathsf{PRF}(k, r^*)$	Security of iO
3	$\mathsf{iO}(\mathsf{Enc}(k_{r^*}, \cdot; \cdot))$	$r^* \leftarrow \{0,1\}^{2\lambda}$	$\boxed{m_b \oplus \text{ Random}}$	Security of PuncPRF Adv $= 0$

Proving Functional Equivalence

We want to show for all m, s and a random r^*:

$$\mathsf{Enc}(k, m; s) = \mathsf{Enc}(k_{r^*}, m; s)$$

Proving Functional Equivalence

We want to show for all m, s and a random r^*:

$$\mathsf{Enc}(k, m; s) = \mathsf{Enc}(k_{r^*}, m; s)$$

Let's expand this:

$$(\mathsf{PRG}(s), \mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m)) = (\mathsf{PRG}(s), \mathsf{PRF}(k_{r^*}, \mathsf{PRG}(s)) \oplus m))$$

Proving Functional Equivalence

We want to show for all m, s and a random r^*:

$$\mathsf{Enc}(k, m; s) = \mathsf{Enc}(k_{r^*}, m; s)$$

Let's expand this:

$$(\mathsf{PRG}(s), \mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m)) = (\mathsf{PRG}(s), \mathsf{PRF}(k_{r^*}, \mathsf{PRG}(s)) \oplus m))$$

This is the case iff:

$$\mathsf{PRF}(k, \mathsf{PRG}(s)) = \mathsf{PRF}(k_{r^*}, \mathsf{PRG}(s))$$

Proving Functional Equivalence

We want to show for all m, s and a random r^*:

$$\mathsf{Enc}(k, m; s) = \mathsf{Enc}(k_{r^*}, m; s)$$

Let's expand this:

$$(\mathsf{PRG}(s), \mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m)) = (\mathsf{PRG}(s), \mathsf{PRF}(k_{r^*}, \mathsf{PRG}(s)) \oplus m))$$

This is the case iff:

$$\mathsf{PRF}(k, \mathsf{PRG}(s)) = \mathsf{PRF}(k_{r^*}, \mathsf{PRG}(s))$$

By the correctness of PuncPRF, this is the case iff:

$$(\forall s) \ : \ \mathsf{PRG}(s) \neq r^*$$

Proving Functional Equivalence

We want to show for all m, s and a random r^*:

$$\mathsf{Enc}(k, m; s) = \mathsf{Enc}(k_{r^*}, m; s)$$

Let's expand this:

$$(\mathsf{PRG}(s), \mathsf{PRF}(k, \mathsf{PRG}(s)) \oplus m)) = (\mathsf{PRG}(s), \mathsf{PRF}(k_{r^*}, \mathsf{PRG}(s)) \oplus m))$$

This is the case iff:

$$\mathsf{PRF}(k, \mathsf{PRG}(s)) = \mathsf{PRF}(k_{r^*}, \mathsf{PRG}(s))$$

By the correctness of PuncPRF, this is the case iff:

$$(\forall s) \ : \ \mathsf{PRG}(s) \neq r^*$$

This holds with overwhelming probability as

- PRG has a sparse image in $\{0,1\}^{2\lambda}$.
- r^* was chosen randomly in $\{0,1\}^{2\lambda}$.

Other Applications of iO

- Deniable Encryption
- Hierarchical/Delegatable Functional Encryption
- Removing Reliance on Random Oracles
- Witness Encryption
- Multilinear Maps
- Zero-Knowledge Proofs
- ...

Further Reading

2001 (Im)possibility of Obfuscating Programs. (Barak et al.)

2007 On Best Possible Obfuscation. (Goldwasser & Rothblum)

2013 Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. (Garg et al.)

2013 How to Use Indistinguishability Obfuscation: Deniable Encryption, and More. (Sahai & Waters)

2014 Cryptographic Obfuscation and "Unhackable" Software. (Blog Post by Matt Green)

2016 Hopes, Fears and Software Obfuscation: A Survey. (Barak)

Further Reading

2001 (Im)possibility of Obfuscating Programs. (Barak et al.)

2007 On Best Possible Obfuscation. (Goldwasser & Rothblum)

2013 Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. (Garg et al.)

2013 How to Use Indistinguishability Obfuscation: Deniable Encryption, and More. (Sahai & Waters)

2014 Cryptographic Obfuscation and "Unhackable" Software. (Blog Post by Matt Green)

2016 Hopes, Fears and Software Obfuscation: A Survey. (Barak)

Thanks.