COS 597C: Recent Developments in Program Obfuscation — Princeton University
Lecture 4 (09/27/16)
Lecturer: Mark Zhandry — Scribe: Jordan Tran

Notes for Lecture 4

1 Last Time

Last time, we started to see some application of indistinguishability obfuscation (iO). In particular, we saw that iO and puncturable pseudo random function (PRF) give us signature. However, we did not learn where puncturable PRF comes from. Basically, we can show that one-way function (OWF) is sufficient to build puncturable PRF. Therefore, we can instead say that OWF and iO give us signature. There is a celebrated result [HILL'99] which showed that OWF implies pseudo random generator (PRG). Today, we are going to show that PRG implies puncturable PRF.

2 Construction of Puncturable PRF

2.1 What is a PRG?

A pseudo random generator (PRG) is a way to stretch out a small source of random bits to a large amount of random looking bits. For our application, we are going to refer to the PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. However, not all PRGs need to be length-doubling. Security requires that the distribution of $\mathsf{G}(s)$ (for $s \leftarrow \{0,1\}^\lambda$) should be computationally indistinguishable from r sampled uniformly random from $\{0,1\}^{2\lambda}$.

2.2 PRF Construction from PRG

We can construct a normal PRF from a length-doubling PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ [GGM'86]. Given an output $\mathsf{G}(s)$, we are going to divide it into two halves $\mathsf{G}_0(s)$ and $\mathsf{G}_1(s)$. First, we define the PRF for a single bit as:

$$\mathsf{PRF}(s,b) = \mathsf{G}_b(s) \qquad\qquad b \in \{0,1\}$$

Then we can recursively define the PRF as:

$$\mathsf{PRF}(s,x||b) = \mathsf{G}_b(\mathsf{PRF}(s,x)) \qquad\qquad b \in \{0,1\}$$

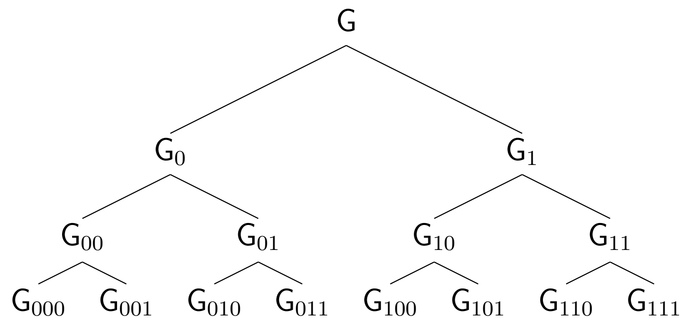

Figure 1: Sample GGM PRF tree with 3 levels

We can observe that the GGM construction of PRF forms a tree (see Figure 1). For n-bit input, we will have n levels of the tree. The leaves of the tree correspond to the output of the PRF. Note that if the input has size λ, the tree is exponentially big. Therefore, we cannot compute the entire tree. However, we can still efficiently compute each particular input to the PRF.

2.3 How to Puncture PRF

Recall that the punctured key for a puncturable PRF allows us to evaluate the PRF tree everywhere but a single point x. Let P_x be the path from root to the leaf x where we want to puncture, and let N_x be the set of neighbors to the nodes in P_x (see Figure 2). Let $\mathcal{K}$ be the PRF key. We can set the punctured key on x, $\mathcal{K}\{x\}$, as the set of values of all the nodes in N_x.

$$\mathcal{K}\{x\} = \{\mathsf{PRF}(\mathcal{K}, y) \mid y \in \mathsf{N}_x\}$$

Note that the values we are giving out with $\mathcal{K}\{x\}$ are all the neighbors to the path to x. This is sufficient to compute the PRF for any value other than x, since we can expand the tree below the neighboring nodes by applying the PRG to the values of the nodes. $\mathcal{K}\{x\}$ can be computed efficiently by keeping all the neighboring values when we compute $\mathsf{PRF}(x)$.

2.4 Security of Construction

For security, we need to argue that given the values of all the neighbors, an adversary still cannot distinguish between the value at the punctured point and a truly random r.

By PRG security, an adversary cannot distinguish between output of the PRG and a truly random value s. We have PRG values at every node in the tree, so we can replace

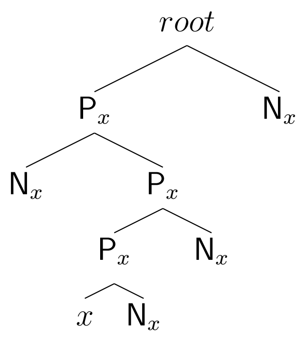

Figure 2: P_x and N_x values for given leaf x in the tree

these values with uniformly random s_0 and s_1 from $\{0,1\}^\lambda$. We can then output s_1 as part of the $\mathcal{K}\{x\}$ and use s_0 to compute x. With s_0 as a truly random value, we can apply PRG security again to replace the children nodes with random values s_{00} and s_{01}. Repeat this process until we arrive at the leaves of the tree. Now we have replaced all the nodes of the tree with truly random values that are independent of each other. Since the value for x is now independent of the rest of the tree, $\mathcal{K}\{x\}$ does not give the adversary any information about x.

3 Public Key Encryption from iO

3.1 Definition

A public key encryption (PKE) scheme consists of three algorithms:

- $\mathsf{Gen}(\lambda)$ takes security parameter λ as input and outputs a pair of keys $(\mathsf{ek}, \mathsf{dk})$, where ek is the public encryption key and dk is the corresponding private decryption key.
- $\mathsf{Enc}(\mathsf{ek}, m)$ is a randomized algorithm that takes the encryption key ek and a message m and outputs a ciphertext C.
- $\mathsf{Dec}(\mathsf{dk}, C)$ takes the decryption key dk and a ciphertext C and outputs a message m.

For correctness, we require that:

$$\mathsf{Dec}(\mathsf{dk}, \mathsf{Enc}(\mathsf{ek}, m)) = m \qquad\qquad (\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}(\lambda)$$

3.2 Semantic Security

The security of a PKE scheme is defined as a game between an adversary and a challenger:

1. The challenger generates $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}(\lambda)$.
2. The challenger sends ek to the adversary.
3. The adversary sends two messages m_0 and m_1 to the challenger.
4. The challenger chooses a random bit b and send $\mathsf{Encrypt}(\mathsf{ek}, m_b)$ to the adversary
5. The adversary outputs guess b'

Let W_b be the event that the adversary outputs 1 when the challenger has the bit b. A PKE scheme is secured if for all PPT adversaries, there exists negligible function negl such that:

$$|Pr[W_0] - Pr[W_1]| \leq \mathsf{negl}(\lambda)$$

3.3 Construction from iO and OWF

We can construct a PKE scheme with OWF and iO as follows:

```
• Gen(λ):
    Generate a random key $\mathcal{K} \leftarrow \{0,1\}^\lambda$. Set $\mathcal{K}$ as the puncturable PRF key.
    Contruct program $\mathsf{P}_\mathcal{K}(s)$ which has the PRF key $\mathcal{K}$ hardcoded.
      $\mathsf{P}_\mathcal{K}(s)$:
        $r \leftarrow \mathsf{PRG}(s)$
        $y \leftarrow \mathsf{PRF}(\mathcal{K}, r)$
        Outputs $(r, y)$
    Set $\mathsf{dk} = \mathcal{K}$ and $\mathsf{ek} = \mathsf{Obf}(\mathsf{P}_\mathcal{K})$ where $\mathsf{Obf}(\mathsf{P}_\mathcal{K})$ is an obfuscation of $\mathsf{P}_\mathcal{K}$.
    Output $(\mathsf{ek}, \mathsf{dk})$

• $\mathsf{Enc}(\mathsf{Obf}(\mathsf{P}_\mathcal{K}), m)$:
    Generate a random $s \leftarrow \{0,1\}^\lambda$
    $(r, y) \leftarrow \mathsf{Obf}(\mathsf{P}_\mathcal{K})(s)$
    Output $(r, y \oplus m)$

• $\mathsf{Dec}(\mathcal{K}, (r, c))$: Output $m = c \oplus \mathsf{PRF}(\mathcal{K}, r)$
```

3.4 Proof of Security

We can prove the security of our construction using a sequence of hybrid experiments, where the changes between two hybrids depend on the security of one of the primitives (PRG, iO, and puncturable PRF).

Let the challenge ciphertext given to the adversary be (r^*, c^*). The series of hybrids is defined as follows:

- H_0: This hybrid corresponds to the security game between the challenger and adversary where the bit b chosen by the challenger is 0.
  $r^* = \mathsf{PRG}(s) \qquad s \leftarrow \{0,1\}^\lambda$
  $y^* = \mathsf{PRF}(\mathcal{K}, r^*)$
  $c^* = y^* \oplus m_0$
- H_1: This hybrid is similar to H_0 except we sample r^* uniformly randomly from $\{0,1\}^\lambda$ instead of using the PRG.
- H_2: In this hybrid, we replace the encryption key in H_1 with $\mathsf{Obf}(\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}})$ (obfuscation of program $\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}}$) where $\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}}$ is defined as follows for hard-coded r^* and punctured PRF key $\mathcal{K}\{r^*\}$:

```
$\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}}(s)$ :
    $r \leftarrow \mathsf{PRG}(s)$
    If $r = r^*$
      Abort and output $\perp$
    Else
      $y \leftarrow \mathsf{PRF}(\mathcal{K}\{r^*\}, r)$
      Output $(r, y)$
```

- H_3: Replace y^* in H_2 with uniformly random value from $\{0,1\}^\lambda$
- H_4: Set $c^* = y^* \oplus m_1$
- H_5: Replace uniformly random y^* in H_4 with $y^* \leftarrow \mathsf{PRF}(\mathcal{K}, r^*)$
- H_6: Set the encryption key back to $\mathsf{Obf}(\mathsf{P}_\mathcal{K})$
- H_7: Replace uniformly random r^* with $r^* \leftarrow \mathsf{PRG}(s)$

Note that H_0 corresponds to the honest security game where the challenger encrypts m_0, and H_7 corresponds to the security game where the challenger encrypts m_1. We now prove that each two consecutive hybrids are indistinguishable from each other.

- H_0 and H_1: If an adversary $\mathcal{A}$ can distinguish between H_0 and H_1, then $\mathcal{A}$ can be used to distinguish $\mathsf{PRG}(s)$ and uniformly random $r \leftarrow \{0,1\}^\lambda$ and break PRG security.

- H_1 and H_2: In hybrid H_2, we replace $\mathsf{Obf}(\mathsf{P}_{\mathcal{K}})$ with $\mathsf{Obf}(\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}})$. Note that in both H_1 and H_2, r^* is chosen uniformly random from $\{0,1\}^\lambda$. However, the PRG has a much smaller domain and range than $\{0,1\}^\lambda$. Therefore, the probability that a uniformly random $r^* \leftarrow \{0,1\}^\lambda$ has a pre-image under the PRG is exponentially small, so with high probability $\mathsf{P}_{\mathcal{K}}$ and $\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}}$ have the same functionality. We can then apply iO security to conclude that $\mathsf{Obf}(\mathsf{P}_{\mathcal{K}}) \approx_c \mathsf{Obf}(\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}})$, and therefore $H_1 \approx_c H_2$.

- H_2 and H_3: Suppose we have adversary $\mathcal{A}$ which can distinguish between H_2 and H_3 with non-negligible advantage. We can then construct an adversary $\mathcal{B}$ on the puncturable PRF using $\mathcal{A}$ as a subroutine:

  1. $\mathcal{B}$ generates random $r^* \leftarrow \{0,1\}^\lambda$ and send to the puncturable PRF challenger.
  2. $\mathcal{B}$ receives punctured key $\mathcal{K}\{r^*\}$ and y^* from puncturable PRF challenger.
  3. $\mathcal{B}$ constructs $\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}}$ and sends $\mathsf{Obf}(\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}})$ as ek to $\mathcal{A}$.
  4. $\mathcal{B}$ receives challenge messages m_0, m_1 from $\mathcal{A}$ and send back $(r^*, y^* \oplus m_0)$ as the challenge ciphertext.
  5. $\mathcal{B}$ outputs guess b' received from $\mathcal{A}$

  If $\mathcal{B}$ receives $y^* = \mathsf{PRF}(\mathcal{K}, r^*)$ from the PRF challenger, the view of $\mathcal{A}$ is identical to H_2. If $\mathcal{B}$ receives truly random y^*, then $\mathcal{A}$'s view corresponds to H_3. Therefore, if $\mathcal{A}$ distinguish between H_2 and H_3, then $\mathcal{B}$ can break punturable PRF security.

- H_3 and H_4: Since y^* is uniformly random, $y^* \oplus m$ for any message m is also uniformly random. Therefore, the distribution of $y^* \oplus m_0$ and $y^* \oplus m_1$ are identical.

- H_4 and H_5: Using puncturable PRF security, we can argue that H_4 and H_5 are also indistinguishable (see $H_2 \approx_c H_3$ proof).

- H_5 and H_6: Indistinguishability between H_5 and H_6 follows from indistinguishability $\mathsf{Obf}(\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}})$ and $\mathsf{Obf}(\mathsf{P}_{\mathcal{K}})$ by iO security (see $H_1 \approx_c H_2$ proof).

- H_6 and H_7: Finally, we can argue that H_6 and H_7 are indistinguishable by PRG security (see $H_0 \approx_c H_1$ proof).

Since consecutive hybrids are all indistinguishable from each other, H_0 is indistinguishable from H_7. H_0 and H_7 correspond to the PKE security game where the challenge coin b is 0 and 1 respectively. Therefore, no PPT adversary can distinguish between encryption of m_0 and encryption of m_1.

We glossed over the fact that in order to use iO, the two programs need to be of the same size. In the proof, we invoked iO on $\mathsf{P}_{\mathcal{K}}$ and $\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}}$. While we proved that they are equivalent, they are not really of the same size. However, this is not a problem for this application, since we can pad the programs to be the same size. Let $S = \mathsf{max}(|\mathsf{P}_{\mathcal{K}}|, |\mathsf{P'}_{r^*,\mathcal{K}\{r^*\}}|)$ (S is efficiently computable since $|r^*|$ is fixed). Before we obfuscate the programs in the hybrids, we can pad them to size S.

4 Multi-Party Non-Interactive Key Exchange

Non-interactive key exchange (NIKE) is a method for multiple parties to, through the minimum amount of interaction, establish a shared secret key, even with the presence of an adversary in the communication channel.

Suppose we have a public bulletin board and parties and parties $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$. Each party is going to compute a secret value sv and a public value pv. All parties then publish their public values $\{\mathsf{pk}_{\mathcal{A}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{D}}\}$ to the board. $\mathcal{A}$ can now compute shared key $\mathcal{K}$ using $(\mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{sk}_{\mathcal{A}})$. $\mathcal{B}$ and other parties should also be able to compute the same key $\mathcal{K}$ from $(\mathsf{pk}_{\mathcal{A}}, \mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{sk}_{\mathcal{B}})$ (or the equivalent tuples for $\mathcal{C}$ and $\mathcal{D}$). Meanwhile, an adversary who see only $\{\mathsf{pk}_{\mathcal{A}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{D}}\}$ should not be able to find $\mathcal{K}$.

The problem is difficult in that all parties compute their values independently, yet they are still able to establish a common key $\mathcal{K}$. In a weaker variant of the scheme with trusted setup, we have a trusted third party who, prior to everything else, publish the public parameter params. All parties can then independently compute their values using params.

For two parties, we can use the Diffie-Hellman key exchange scheme [DH'76]. Using bilinear maps, we can expand the key exchange to three parties. In the next lecture, we will how to obtain NIKE for more than three parties with no trusted setup.

References

[DH'76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644- 654, 1976. 7

[GGM'86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792- 807, 1986. 1

[HILL'99] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way funtion. *SIAM Journal on Computing*, 28(4):1364- 1396, 1999. 1