INDISTINGUISHABILITY OBFUSCATION

SAMARTH DAS

Abstract. This paper discusses various applications of indistinguishability obfuscation to cryptographic problems and their constructions.

1. What is Program Obfuscation?

You may be familiar with a password, which is a secret key that only you know that can give you access to various different things such as your email account. When you enter the wrong password, the system informs you of this because it is aware of the correct one and prevents an attacker from accessing your account. But what if we were able to keep a secret information that the software and the person who runs the software also does not know? This is called program obfuscation.

2. Indistinguishability Obfuscation

The simplest way to do so is called **indistinguishability obfuscation**. This system says that if we have two equivalent programs P_1 and P_2 where $(P_1(x) = P_2(x) \forall x$, and one of these is obfuscated, then it is impossible to tell which one is obfuscated. To make this definition rigorous, we have the following:

Definition 2.1 (Indistinguishability Security for Obfuscation)**.** An obfuscator $\texttt{Obf}$ is indistinguishability secure if the following holds. For any two programs P_0, P_1 that are equivalent $(P_1(x) = P_2(x) \forall x$ and have the same size, for any PPT adversary A, there exists a negligible function $\texttt{negl}$ such that

$$\Pr[A(\texttt{Obf}(P_0)) = 1] - \Pr[A(\texttt{Obf}(P_1)) = 1] < \texttt{negl}(\lambda)$$

It turns out with this definition we can apply iO (the short form for indistinguishability obfuscation) to a wide variety of cryptographic tasks as we will see.

3. Digital Signatures

Consider the two people Alice and Bob. They want to communicate in such a manner that any adversary trying to intercept and modify the message can be detected.

We can do this with the use of a digital signature scheme where Alice generates a public verification key $\texttt{vk}$ and gives it to Bob, and generates for herself a private signing key $\texttt{sk}$. When Alice sends a message m, she uses her signing key $\texttt{sk}$ to generate a signature, and forwards this along with m. If Bob receives $m' = m$, we want him to reject the message.

The signature method is as follows:

- $\texttt{Gen}(\lambda)$ takes the security parameter λ as input and outputs ($\texttt{sk}$,$\texttt{vk}$).
- $\texttt{Sign}(\texttt{sk}, m)$ takes the secret key $\texttt{sk}$ and the message m as input and outputs σ.

Date: August 19, 2024.

- $\texttt{Ver}(\texttt{vk}, m, \sigma)$ takes the verification key $\texttt{vk}$, the message m, and and outputs accept or reject.

The obvious correctness requirement is that if we start with a message m and generate $\texttt{Sign}(\texttt{sk}, m) \to \sigma$, then $\texttt{Ver}(\texttt{vk}, m)$ accepts. The natural way to attack this is a chosen message attack, where the adversary commits to a message m' and sends it to the challenger. The challenger generates the secret key/verification key pair $\texttt{(sk,vk)} \leftarrow \texttt{Gen}(\lambda)$ and sends $\texttt{vk}$ back to the adversary. At this point, the adversary may make polynomially many queries message queries to the challenger, where he sends a message $m' \neq m$ and receives $\texttt{Sign}(\texttt{sk}, m)$. The adversary generates a new signature σ' and wins if $\texttt{Ver}(\texttt{vk}, m', \sigma') = accept$. The adversaries advantage is his probability of winning.

To make this secure, we introduce a variant of the pseudorandom function PRF, a punctured PRF:

Definition 3.1 (Puncturable PRF)**.** A puncturable PRF is a function $PRF(k, x) \leftarrow y$ which has an algorithm $k\{x'\} \leftarrow \texttt{Puncture}(k, x')$ that outputs a punctured key.

Then we require that $(k\{x'\}, PRF(k, x')) \approx_c (k\{x'\}, R)$ where R is a random value. This means that the value at x' of the PRF is indistinguishable from a random value R.

Construction 3.2. We can make the following working construction:

- $\texttt{Gen}(\lambda)$ samples $\texttt{sk} = k \leftarrow \{0,1\}^\lambda$. Define $C_k(m, \sigma) = F(PRF_k(m)) = F(\sigma)$ where F is a one way function. Then we set $\texttt{vk} = O(C_k) = \hat{C}$.
- $\texttt{Sign}(k, m) = PRF(k, m)$
- $\texttt{Ver}(\hat{C}, m, \sigma) = \hat{C}(m, \sigma)$

4. Public Key Encryption

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

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

For correctness, we require that: $\texttt{Dec(dk,Enc(ek}, m)) = m$ and $\texttt{(ek,dk)} \leftarrow \texttt{Gen}(\lambda)$.

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

(1) The challenger generates $\texttt{(ek,dk)} \leftarrow \texttt{Gen}(\lambda)$.
(2) The challenger sends $\texttt{ek}$ 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 $\texttt{Encrypt}(\texttt{ek}, m_b)$ 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.

Definition 4.1 (PKE security)**.** A PKE scheme is secured if for all PPT adversaries, there exists a negligible function $\texttt{negl}$ such that: $|Pr[W_0] - Pr[W_1]| \leq \texttt{negl}(\lambda)$

Construction 4.2. We can construct a $\texttt{PKE}$ scheme with $\texttt{OWF}$ and $\texttt{iO}$ as follows:

- $\texttt{Gen}(\lambda)$: Generate a random key $\mathcal{K} \leftarrow \{0,1\}^\lambda$. Set $\mathcal{K}$ as the puncturable $\texttt{PRF}$ key. Construct program $\texttt{P}_\mathcal{K}(s)$ which has the $\texttt{PRF}$ key $\mathcal{K}$ hardcoded. $\texttt{P}_\mathcal{K}(s)$: $r \leftarrow \texttt{PRG}(s)$ $r \leftarrow \texttt{PRF}(\mathcal{K}, r)$ Outputs (r, y) Set $\texttt{dk} = \mathcal{K}$ and $ek = \texttt{Obf}(P - \mathcal{K})$ where $\texttt{Obf}(P - \mathcal{K})$ is an obfuscation of $\texttt{P}_\mathcal{K}$. Output $\texttt{(ekdk)}$
- $\texttt{Enc(Obf(P}_\mathcal{K}\texttt{)}, m)$: Generate a random $s \leftarrow \{0,1\}^\lambda$ $(r, y) \leftarrow \texttt{Obf(P}_\mathcal{K}\texttt{)}(s)$ Output $(r, y \bigoplus m)$
- $\texttt{Dec}(\mathcal{K}(r, c))$: Output $m = x \bigoplus \texttt{PRF}(\mathcal{K}, r)$

.

5. Multiparty 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 $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$. Each party is going to compute a secret value $\texttt{sv}$ and a public value $\texttt{pv}$. All parties then publish their public values $\texttt{pk}_A, \texttt{pk}_B, \texttt{pk}_C, \texttt{pk}_D$ to the board. A can now compute shared key $\mathcal{K}$ using $\texttt{pk}_B, \texttt{pk}_C, \texttt{pk}_D, \texttt{sk}_A$. B and other parties should also be able to compute the same key $\mathcal{K}$ from $\texttt{pk}_A, \texttt{pk}_C, \texttt{pk}_D, \texttt{sk}_B$ or the equivalent tuples for $\mathcal{C}$ and $\mathcal{D}$. Meanwhile, an adversary who see only $\texttt{pk}_A, \texttt{pk}_B, \texttt{pk}_C, \texttt{pk}_D$ 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 $\texttt{params}$. All parties can then independently compute their values using $\texttt{params}$. For two parties, we can use the Diffie-Hellman key exchange scheme. Using bilinear maps, we can expand the key exchange to three parties.

Now we look at how we can have a shared key. Alice and Bob have a channel through which they can communicate over many rounds. The goal is for Alice and Bob to agree on a shared key k that an eavesdropper (who can see every round of communication) is unable to deduce anything about. This two party system is easily solved by the public key encryption we described earlier.

Why is non-interactive key exchange particularly interesting? It is interesting to see if we can compress communication to just a single, non interactive round. The messages/public values can be reused. The fact that this procedure is non interactive means that Alice and Bob can post their public values to establish a shared key, and later, if Charlie wants to establish a shared key with just Bob, Bob doesnt have to post any new messages. Charlie just runs his half of the protocol, and posts his public value. Then Bob can compute the shared key k_{BC} right away

Definition 5.1 (Multiparty NIKE)**.** We define a multiparty NIKE scheme to consist of the following PPT algorithms:

- $\texttt{Setup}(\lambda, N) \rightarrow \texttt{params}$
- $\texttt{Publish}, (\texttt{params}, i) \rightarrow (sv_i, pv_i)$
- $\texttt{KeyGen}(\texttt{params}, \{pv_i\}_{n=1,\ldots,N}, sv_j) \rightarrow k$

To make a proof of security work with just iO, we need to make a few changes. We turn the one way function into a PRG, and the PRF into a puncturable PRF. The PRG will be length doubling: $\{0,1\}^\lambda \rightarrow \{0,1\}^{2\lambda}$.

Construction 5.2. Then we make the following construction:

- $\texttt{Setup}(\lambda, N)$ samples a totally random $k_PRF \leftarrow \{0,1\}^\lambda$,and outputs $iO(P_{k_PRF}, \lambda, N)$, where P_{k_PRF}, λ, N is as defined above (with the given modifications).
- $\texttt{Publish}(\hat{P})$, sets $sv \leftarrow \{0,1\}^\lambda, pv \leftarrow PRG(sv)$, and then outputs pv, sv.
- $\texttt{KeyGen}(\hat{P}, \{pv_i\}_{n=1,\ldots,N}, i, sv_i) \rightarrow k$ outputs $\hat{P}(\{pv_i\}, i, sv_i)$.

6. Conclusion

Obfuscation is a very challenging task. Indeed, someone who posses the program has many ways to try to extract information. They can run the program on inputs of their choice, and get to see all intermediate states of the program in addition to the program output. They can even inject faults changing the internal state and observing how that affects program behavior. They can do all of this, and yet still should not be able to learn anything about the implementation details of the program.

References

[1] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *Journal of the ACM (JACM)*, 65(6):1–37, 2018.

[2] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[3] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 736–749, 2021.

[4] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

[5] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 475–484, 2014.

[1] [2] [3] [4] [5]