4 A Distributed Pseudo-Random Function

A *pseudo-random function (PRF)* $F_x : \{0,1\}^* \to \{0,1\}^k$ is parameterized by a secret key x (called the *seed*) and maps an arbitrary-length input string to a fixed-length, k-bit output string that looks random to anyone who does not know the secret key. More formally, the PRF is secure if any efficient adversary who queries an oracle with distinct inputs cannot tell, with better than negligible probability, whether the oracle responds to the queries by evaluating the PRF on a random seed (known only to the oracle) or whether the oracle responds every time with a k-bit string freshly chosen at random with uniform distribution [Gol04].

In practice one implements a PRF by a block cipher with a secret key; distributed implementations, however, are only known for functions based on public-key cryptosystems. Cachin et al. [CKS05] describe the following *threshold PRF*, which is suitable for integration in distributed protocols.

Algorithm 1 ([CKS05]). Recall the discrete-logarithm setting with $G = < g >$; let x be a randomly chosen *seed* and define a $F_x : \{0,1\}^* \to \{0,1\}^k$ as

$$F_x(v) \;=\; H'\big(H(v)^x\big),$$

where $H : \{0,1\}^* \to G$ and $H' : G \to \{0,1\}^k$ are two hash functions. The family $F = \{F_x\}$ is a pseudorandom function, assuming the hardness of the DLP (which can proven formally when H is modeled as a so-called random oracle).

A *threshold PRF* can be obtained analogously to threshold ElGamal encryption. Let a trusted dealer choose the seed x and share it among the servers using a polynomial of degree t, such that server P_i holds share x_i. When it is time to compute $F_x(v)$, every correct server P_i computes a function share $d_i = \big(H(v)\big)^{x_i}$ and releases d_i according to the protocol. Any $t+1$ correctly computed function shares, from servers with indices in a set $\mathcal{S}$, yield the value of the PRF,

$$F_x(v) \;=\; H'\Big(\prod_{i\in\mathcal{S}} {d_i}^{\lambda_{0,i}^{\mathcal{S}}}\Big).$$

Writing $h = H(v)$, this is correct because

$$\prod_{i\in\mathcal{S}} {d_i}^{\lambda_{0,i}^{\mathcal{S}}} \;=\; \prod_{i\in\mathcal{S}} h^{x_i\lambda_{0,i}^{\mathcal{S}}} \;=\; h^{\sum_{i\in\mathcal{S}} x_i\lambda_{0,i}^{\mathcal{S}}} \;=\; h^x.$$

One can show that this does not leak information about x, under the assumption that the DLP is hard.

The threshold PRF can replace a sequence of shared coins $s_0, s_1, \ldots$ in a randomized Byzantine consensus protocol like Algorithm 10 of Chapter 3 (*Byzantine Broadcasts and Randomized Consensus*). Concretely, one sets output length $k = 1$ and lets the input string for coin s_r consist of $ID\|r$, where ID denotes a unique *tag* identifying the protocol instance that must also be contained in every message and included in all signatures, and where r denotes the index of the coin.

The threshold pseudorandom function is non-interactive. This means that no interaction among the servers is needed to compute the function value. To implement the *recover* operation for s_r in Algorithm 10 of Chapter 3, every server sends its function share of s_r to all others, collects $t + 1$ shares, and combines them to the coin value $F_x(ID\|r)$.

The threshold PRF as described here tolerates only a passive adversary, but one can easily make it robust against an active adversary by adding zero-knowledge proofs for the correctness of the function shares generated by the servers [CKS05].

References

[CKS05] C. Cachin, K. Kursawe, and V. Shoup, *Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography*, Journal of Cryptology **18** (2005), no. 3, 219–246.

[Gol04] O. Goldreich, *Foundations of cryptography*, vol. I & II, Cambridge University Press, 2001–2004.