.1

CS276 Cryptography **Spring 2004**

Lecture 17.1 : 3.29.04

Lecturer: David Wagner *Scribe: Jacob Scott*

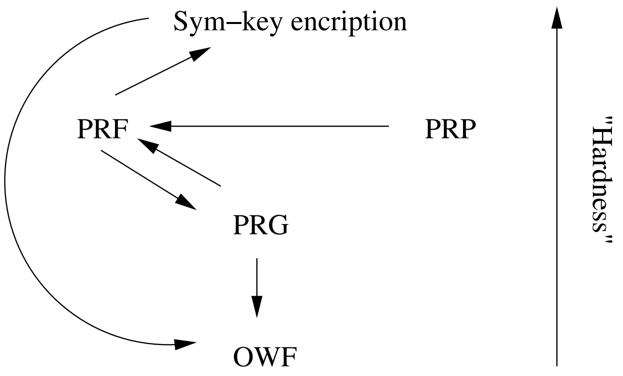

Figure 17.1: Implications between crypto primitives

By this point in the semester, most of the foundations and building blocks of cryptography have been covered. In this lecture the necessary assumptions (e.g., that $P \neq NP$) for these constructions to be secure will be examined along with the relationships and implications between them. Example: it has been shown in previous lectures that PRF $\rightarrow$ symmetric key encryption; that given a secure PRF, it can be used in conjunction with CTR mode to construct a secure symmetric key encryption system. **Unless otherwise noted, the security being discussed is asymptotic**

17.1 OWF Definition

A **O**ne **W**ay **F**unction is one a function f that can be computed in reasonable (polynomial) time, but for which it is hard (hopefully, not polynomial time) to invert - that is, it is not feasible to find an input to f that will produce a specified output. UNIX password hashing is an example of an OWF. Formally:

Definition 17.1 $f : \{0,1\}^m \rightarrow \{0,1\}^n$ *is an OWF if*

$$\forall A,\ Pr_{x \in \{0,1\}^m}[f(A(f(x))) = f(x)] \leq \epsilon$$

Since f is not necessarily bijective, this says that given $f(x)$, A should not be able to return any x' such that $f(x') = f(x)$.

17.2 Showing Implications

Note in addition to PRF → symmetric key encryption, PRP → PRF was proved previously in class using the birthday paradox.

17.2.1 PRG → OWF

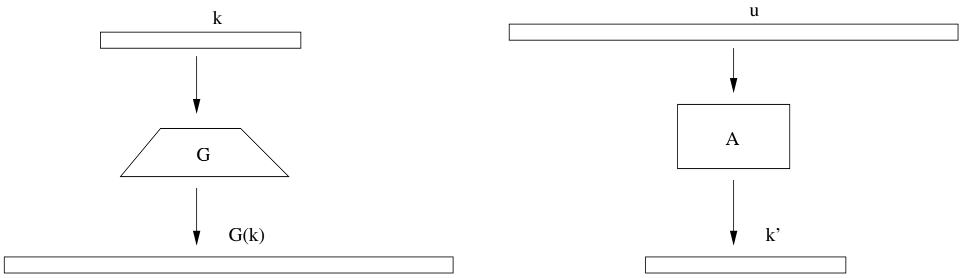

Figure 17.2: Operation of G and A attempting to invert random strings

Consider some PRG $G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$. Then G is **already** an OWF.

Informal Proof: Consider any algorithm A which OWF-breaks G. By definition, A is good at inverting G. However, if A is given inputs U.A.R rather than those generated by G, it will not be good at inverting them (e.g., it is unlikely for A to find inputs to G that map to arbitrary random strings of length 2^{2n}). Thus, A can be used to distinguish between the pseudo-random and true random, and PRG-break G. Since G is a PRG by definition, no such A can exist.

Because G is deterministic, it uses only an exponentially sparse subset of its range - each input of length 2^n maps to one output of length 2^{2n} - so G has at most 2^n distinct outputs, and there do not exist inputs for G that will result in the output of the other $2^{2n} - 2^n$ bit strings in its range. Since a random bit string of length 2^{2n} will be one which is not a possible output of G with high probability, the probability that A fails to generate a valid input to G (which in this case is impossible) is also high.

17.2.2 Sym-key → OWF

An OWF f can be constructed from an encryption system E as follows:

$$f(x) = E_x(something)$$

If f is inverted, then key recovery has occurred. So if E is secure as a symmetric key encryption system, f must be secure as an OWF.

17.2.3 PRF → PRG

A PRG $G : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ can be constructed from a PRF $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ as follows:

$$G(k) = (F_k(0), F_k(1))$$

This is basically using F in CTR mode, except for the stage where the plaintext is XORed in. The intuition is that both PRGs and CTR mode encryption are stream ciphers, so CTR mode can be used to build the PRG.

17.2.4 PRG → PRF

Throughout this section let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a length doubling PRG. Furthermore, Let G_0 be the first n bits of the output of G and G_1 be the second n bits, so that $G(k) = (G_0(k), G_1(k))$

17.2.4.1 1-bit PRF

A 1-bit PRF $F : \{0,1\}^n \times \{0,1\} \to \{0,1\}^n$ can be constructed from G as follows:

$$F_k(b) = G_b(k)$$

Since G outputs 2^{2n} pseudo-random bits, simply return the first or second half of them based on the bit input to F.

17.2.4.2 l-bit PRF

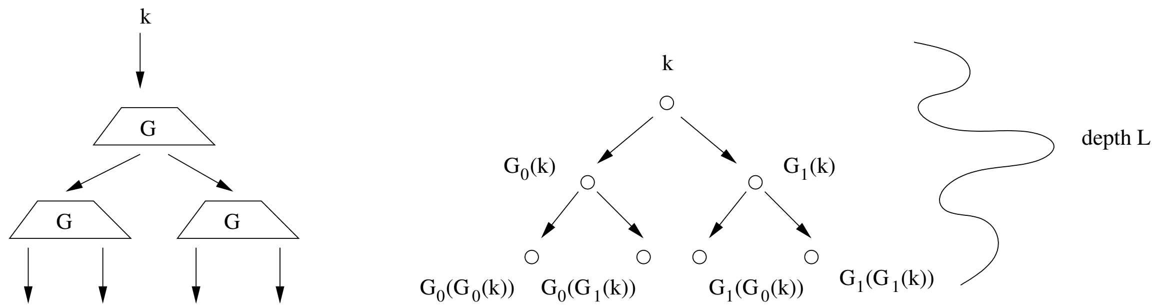

Figure 17.3: Tree of calls to G whose leaves are the possible outputs of F

A l-bit PRF $F' : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^n$ can be made in the same way from G (known as the GGM construction):

$$F_k(m_1 m_2 ... m_l) = G_{m_1}(G_{m_2}(G_{m_2}(\ldots G_{m_l}(k))))$$

This construction makes use of 2^l pseudo-random bits, which is bad because it was shown in HW2 (problem 2, on constructing large PRGs) that this will cause an attacker's advantage to scale exponentially.

Luckily, a shift to concrete security can help. In (t, q, ϵ)-security, only q queries, or calls to F, can be made (the attacker can only see q leaves of the tree in Figure 17.3). Since the leaf size is polynomial, it can be shown that seeing q leaves will give any attacker an advantage no greater than $q\epsilon$, for the underlying PRF's ϵ.