CS 276 Lecture Notes

Ben Schwarz (bschwarz@cs.berkeley.edu)

April 5, 2004

1 Implications

We have seen a number of implications that prove the existence of important cryptographic primitives. For instance, we know how to build a coin flipping protocol using bit-commitment, and vice-versa. The following diagram shows the implications we have seen.

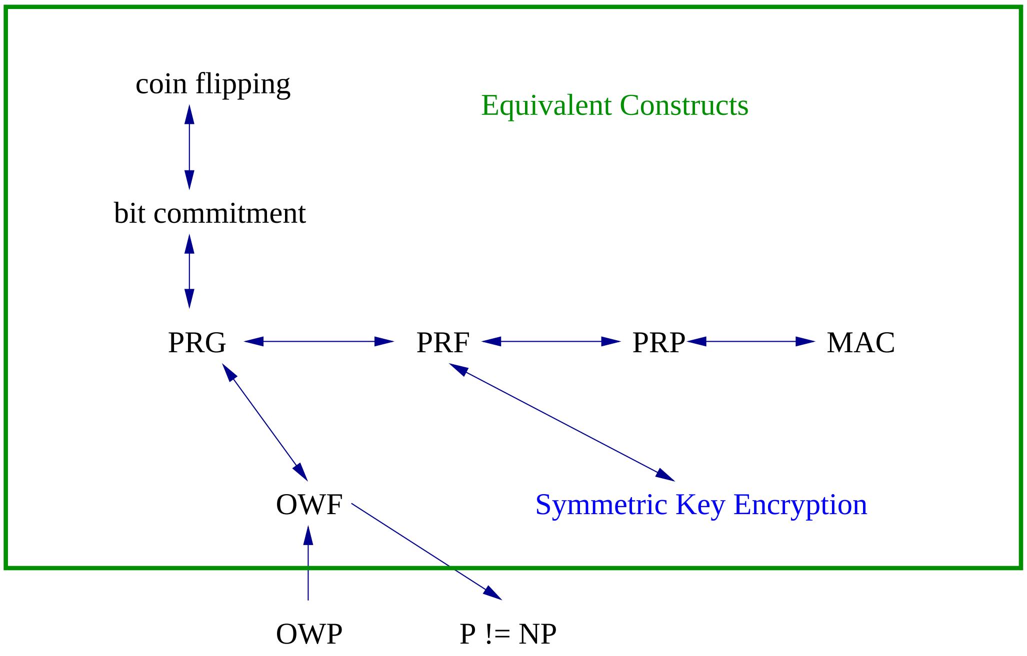

Figure 1: Implications in the symmetric key world

We have also seen some implications in the public key setting. They are shown in Figure 2.

For instance, to show public key encryption implies public key agreement, we can construct a protocol by which two parties choose public and private key pairs, set up a secure channel, then agree on a symmetric key.

We may be tempted to try to draw arrows between the symmetric key and the public key worlds using black box reductions. However, we there is no known implication from

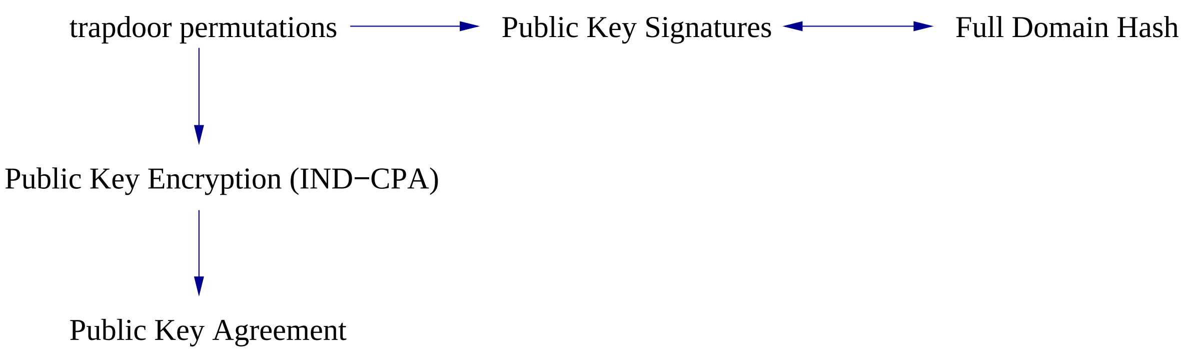

Figure 2: Implications in the public key world

one way functions to public key encryption using this technique. In fact, such a reduction is impossible (Impagliazzo-Rudich).

2 Blackbox Reductions and Separations

We have seen blackbox reductions in many proofs thus far. We can formalize the notion as follows.

Suppose we want to build a Q out of a P. Let $\mathtt{I}$ be a secure implementation of P. Then $\mathtt{J}^I$ is a secure Q that can issue queries to its oracle, $\mathtt{I}$. In addition, if we know to how attack $\mathtt{J}^I$ with an algorithm B, then there exists an algorithm A^B that attacks $\mathtt{I}$ using B as its oracle. We have seen the latter used in many scenarios–for example, showing left-or-right security is equivalent to real-or-random.

Surprisingly, it is possible to show that there exists no blackbox reduction from public key encryption to one-way functions. This result comes from Impagliazzo and Rudich. Given that all but one of the major cryptographic results using reduction from one construct to another has been blackbox reduction, the authors conjecture it will be extremely difficult to ever prove OWF implies public key crypto.

3 Building Signatures from One-Way Functions

We have seen how to build signature schemes from cryptographic constructs in the public key world. But can we do it something from the symmetric key world? The answer is yes. It is possible to use one-way functions (and consequently anything else that is equivalent in Figure 1). to create signature schemes. The implementation seen below is particularly inefficient; for an n-bit message we are required to perform $O(n)$ hashes.

We are going to construct a private and public keypair. To transmit an n-bit message, the keypairs are going to each be 2* 128 bits * n in length. The private keypair is chosen to be a concatenation of random numbers:

$$r_{i,j} \leftarrow \{0,1\}^n \ \forall \text{ i from 1...n, j from \{0,1\}}$$
$$\text{tk} \leftarrow r_{1,0} \circ r_{1,1} \circ \ldots\ r_{n,0} \circ r_{n,1}$$

The public key is then constructed to be the concatenation of the hash of each of the inidivdual $r_{i,j}$'s:

$$\text{pk} \leftarrow \text{h}(r_{1,0}) \circ \text{h}(r_{1,1}) \circ \ldots\ \text{h}(r_{n,0}) \circ \text{h}(r_{n,1})$$

To send an n-bit message, we look at the i'th bit of the message. If it is 0, we use $r_{i,0}$. If it is 1 we use $r_{i,1}$.

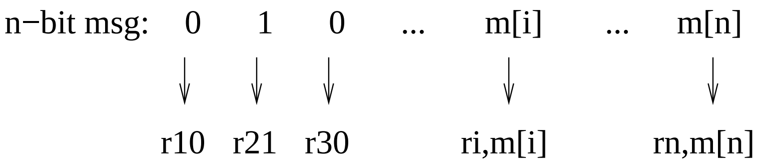

Figure 3: Signatures using a OWF (hash)

The signature of a message m is then the concatenation of the $\mathtt{r}$'s. Intuitively we think it should be secure because the hash function cannot be inverted—we can find no pre-image. Note that it is not secure against multi-query attackers though. You can sign m and then the complement of m, and learn the secret key. One way to reduce the size of the keypair is to use a Merkle tree as shown in Figure 4.

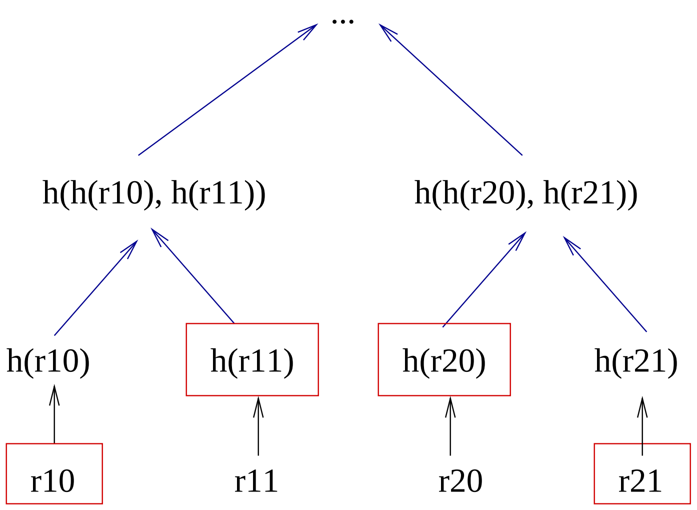

Figure 4: Using a Merkle Tree to shorten public key

The tree is construct bottom-up by hashing the value of the two children at each node, until we come to the root node. Then, the single root value is output as the public key. The recipient then learns half of the leaves of the tree (the ones shown in red). This scheme can be extended so we can do multiple messages as well. Assume S is a 1-time signature scheme. Then to get a 2-time signature scheme, we construct a new S:

$$S'_{tk} = (pk_1,\, pk_2,\, S_{tk}(pk_1,pk_2),\, S_{tk1}(m_1),\, S_{tk2}(m_2))$$

We choose two public and private keypairs. On the first message query, we output

$$(pk_1,\, pk_2,\, S_{tk}(pk_1,\, pk_2),\, S_{tk1}(m_1)).$$

On the second message query we output

$$(pk_1,\, pk_2,\, S_{tk}(pk_1,\, pk_2),\, S_{tk2}(m_2)).$$

To verify the signature scheme, we accept if

$$V_{pk1}(\text{sig(M)}) = \text{M or } V_{pk2}(\text{sig(M)}).$$