1 Introduction to These Notes

These notes are intended to supplement, not replace, the lectures given in class. In particular, only the technical aspects of the lecture are reproduced here; much of the surrounding discussion that took place in class is not.

2 Trapdoor Permutations

We give two definitions of trapdoor permutations. The first is completely formal, and maps well onto the (conjectured) trapdoor permutations that are used in practice. The second is slightly less formal, but is simpler to use and somewhat easier to understand. Generally speaking, however, proofs of security using the second of the two definitions can be easily modified to work for the first definition as well. (Stronger definitions of trapdoor permutations are sometimes also considered; see [1, 2] for some examples.)

We begin with a syntactic definition, and then give a concrete example. Before giving the definition we introduce two pieces of notation. First is the abbreviation PPT which stands for "probabilistic, polynomial-time". This, of course, refers to algorithms which may make random choices during their execution but which always terminate in a polynomial number of steps. This begs the following question: polynomial in *what*? To deal with this, we introduce the notion of a *security parameter* k which will be provided as input to all algorithms. For technical reasons, the security parameter is given in unary and is thus represented as 1^k. In some sense, as we will see, a larger value of the security parameter results in a "more secure" scheme. (Hopefully, the concrete example that follows will give some more motivation for the purpose of the security parameters.)

Definition 1 A *trapdoor permutation* family is a tuple of PPT algorithms (Gen, Sample, Eval, Invert) such that:

1. $\mathsf{Gen}(1^k)$ is a probabilistic algorithm which outputs a pair (i, td). (One can think of i as indexing a particular permutation f_i defined over some domain D_i, while td represents some "trapdoor" information that allows inversion of f_i.)

2. $\mathsf{Sample}(1^k, i)$ is a probabilistic algorithm which outputs an element $x \in D_i$ (assuming i was output by Gen). Furthermore, x is uniformly distributed in D_i. (More formally, the distribution $\{\mathsf{Sample}(1^k, i)\}$ is equal to the uniform distribution over D_i.)

3. $\mathsf{Eval}(1^k, i, x)$ is a deterministic algorithm which outputs an element $y \in D_i$ (assuming i was output by Gen and $x \in D_i$). Furthermore, for all i output by Gen, the function $\mathsf{Eval}(1^k, i, \cdot) : D_i \to D_i$ is a permutation. (Thus, one can view $\mathsf{Eval}(1^k, i, \cdot)$ as corresponding to a permutation f_i mentioned above.)

4. $\mathsf{Invert}(1^k, \mathsf{td}, y)$ is a deterministic algorithm which outputs an element $x \in D_i$, where (i, td) is a possible output of Gen.

Furthermore, we require that for all k, all (i, td) output by Gen, and all $x \in D_i$ we have $\mathsf{Invert}(1^k, \mathsf{td}, \mathsf{Eval}(1^k, i, x)) = x$. (This is our **correctness** requirement.) $\diamondsuit$

The correctness requirement enables one to associate $\mathsf{Invert}(1^k, \mathsf{td}, \cdot)$ with f_i^{-1}. However, it is crucial to recognize that while, as a mathematical function, f_i^{-1} always exists, this function is not necessarily efficiently computable. The definition above, however, guarantees that it *is* efficiently computable, given the "trapdoor" information td (we will see below that, informally, if the trapdoor permutation family is secure then f_i^{-1} is *not* efficiently computable *without* td).

Before going further, we give as a concrete example one of the most popular trapdoor permutations used in practice: RSA [3] (some basic familiarity with RSA is assumed; we merely show how RSA fits into the above framework).

1. $\mathsf{Gen}(1^k)$ chooses two random, k-bit primes p and q, and forms their product $N = pq$. It then computes $\varphi(N) = (p-1)(q-1)$, chooses e relatively prime to $\varphi(N)$, and computes d such that $ed = 1 \bmod \varphi(N)$. Finally, it outputs $((N, e), (N, d))$ (note that i in the definition above corresponds to (N, e) while td corresponds to (N, d)). The domain $D_{N,e}$ is just $\mathbb{Z}_N^*$. (We can also see how the security parameter k comes into play: it determines the length of the primes making up the modulus N, and thus directly affects the "hardness" of factoring the resulting modulus.)

2. $\mathsf{Sample}(1^k, (N, e))$ simply chooses a uniformly-random element from $\mathbb{Z}_N^*$. We noted in class that it is possible to do this efficiently.

3. $\mathsf{Eval}(1^k, (N, e), x)$, where $x \in \mathbb{Z}_N^*$, outputs $y = x^e \bmod N$.

4. $\mathsf{Invert}(1^k, (N, d), y)$, where $y \in \mathbb{Z}_N^*$, outputs $x = y^d \bmod N$.

It is well-known that Invert indeed computes the inverse of Eval. Hence, RSA (as described above) is a trapdoor permutation family.

2.1 Trapdoor (One-Way) Permutations

The definition above was simply syntactic; it does not include any notion of "hardness" or "security". However, when cryptographers talk about trapdoor permutations they always mean *one-way* trapdoor permutations. Informally, this just means that a randomly-generated trapdoor permutation is hard to invert when the trapdoor information td is not known. In giving a formal definition, however, we must be careful: what do we mean by "hard to invert"? Roughly speaking, we will say this means that any "efficient" algorithm succeeds in inverting a randomly-generated f_i (at a random point) with only "very small" probability. (Note that it only makes sense to talk about the hardness of inverting a randomly-generated trapdoor permutation. If we *fix* a trapdoor permutation f_i then it may very well be the case that an adversary knows the associated trapdoor. A similar argument shows that the point to be inverted must be chosen at random as well.) It should be no surprise that we associate "efficient" algorithms with PPT ones. Our notion of "small" is made precise via the class of *negligible functions*, which we define now.

Definition 2 A function $\varepsilon : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\}$ is *negligible* if it is asymptotically smaller than any inverse polynomial. More formally, this means that for all $c > 0$ there exists an integer N_c such that:

$$N > N_c \Rightarrow f(N) < 1/N^c.$$

$\diamondsuit$

We will now formally define the notion of being hard to invert, and thus formally define the notion of one-way trapdoor permutation families.

Definition 3 A trapdoor permutation family $(\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval}, \mathsf{Invert})$ is *one-way* if for any PPT A the following is negligible (in k):

$$\Pr[(i, \mathsf{td}) \leftarrow \mathsf{Gen}(1^k); y \leftarrow \mathsf{Sample}(1^k, i); x \leftarrow A(1^k, i, y) : \mathsf{Eval}(1^k, i, x) = y]. \tag{1}$$

$\diamondsuit$

A few words are in order to explain Eq. (1), especially since this notation will be used extensively throughout the rest of the semester. The equation represents the probability of a particular event following execution of a particular experiment; the experiment itself is written to the left of the colon, while the event of interest is written to the right of the colon. Furthermore, individual components of the experiment are separated by a semicolon. We use the notation "$\leftarrow$" to denote a randomized procedure: if S is a set, then "$x \leftarrow S$" denotes selecting x uniformly at random from S; if A is a randomized algorithm, then "$x \leftarrow A(\cdots)$" represents running A (with uniformly-chosen randomness) to obtain output x. Finally, in an *experiment* (i.e., to the left of the colon) "=" denotes assignment (thus, e.g., if A is a *deterministic* algorithm then we write $x = A(\cdots)$); on the other hand, in an event (i.e., to the right of the colon), "=" denotes a test of equality.

Thus, we may express Eq. (1) in words as follows:

> The probability that $\mathsf{Eval}(1^k, i, x)$ is equal to y upon completion of the following experiment: run $\mathsf{Gen}(1^k)$ to generate (i, td), run $\mathsf{Sample}(1^k, i)$ to generate y, and finally run $A(1^k, i, y)$ to obtain x.

Note also that Eq. (1) is indeed a function of k, and hence it makes sense to talk about whether this expression is negligible or not.

From now on, when we talk about "trapdoor permutations" we always mean "a one-way trapdoor permutation family".

2.2 A Simplified Definition of Trapdoor Permutations

The above definition is somewhat cumbersome to work with, and we therefore introduce the following simplified definition. As noted earlier, this definition does not map well (and sometimes does not map at all) to the trapdoor permutations used in practice; yet, proofs of security using this definition are (in general) easily modified to hold with regard to the more accurate definition given above. (Of course, when giving proofs of security based on trapdoor permutations, one should always be careful to make sure that this is the case.)

The following definition introduces two simplifying assumptions and one notational simplification: our first assumption is that all D_i are the same for a given security parameter

k. Furthermore, for a given security parameter k we will simply assume that $D_i = \{0,1\}^k$ (i.e., the set of strings of length k). We simplify the notation as follows: instead of referring to an index i and a trapdoor td, we simply refer to a permutation f and its inverse f^{-1}. (Technically, one should think of f as a *description* of f, which in particular allows for efficient computation of f; analogously, one should think of f^{-1} as a description of (an efficient method for computing) f^{-1}. In particular, it should always be kept in mind that the mathematical function f^{-1} will not, in general, be computable in polynomial time without being given some "trapdoor" information (which we are here representing by "f^{-1}").)

Definition 4 A trapdoor permutation family is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ such that:

1. $\mathsf{Gen}(1^k)$ outputs a pair (f, f^{-1}), where f is a permutation over $\{0,1\}^k$.
2. $\mathsf{Eval}(1^k, f, x)$ is a deterministic algorithm which outputs some $y \in \{0,1\}^k$ (assuming f was output by Gen and $x \in \{0,1\}^k$). We will often simply write $f(x)$ instead of $\mathsf{Eval}(1^k, f, x)$.
3. $\mathsf{Invert}(1^k, f^{-1}, y)$ is a deterministic algorithm which outputs some $x \in \{0,1\}^k$ (assuming f^{-1} was output by Gen and $y \in \{0,1\}^k$). We will often simply write $f^{-1}(y)$ instead of $\mathsf{Invert}(1^k, f^{-1}, y)$.
4. (**Correctness.**) For all k, all (f, f^{-1}) output by Gen, and all $x \in \{0,1\}^k$ we have $f^{-1}(f(x)) = x$.
5. (**One-wayness**.) For all PPT A, the following is negligible:

$$\Pr[(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k); y \leftarrow \{0,1\}^k; x \leftarrow A(1^k, f, y) : f(x) = y].$$

$\diamondsuit$

Given the above notation, we can just as well associate our trapdoor permutation family with Gen (and let the algorithms Eval and Invert be entirely implicit).

References

[1] O. Goldreich. *Foundations of Cryptography, vol. 1: Basic Tools*, Cambridge University Press, 2001.

[2] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*, Cambridge University Press, to appear.

[3] R. Rivest, A. Shamir, and L.M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM* 21(2): 120–126 (1978).