1 Introduction to Public Key Encryption

Diffie-Hellman’s paper of ’76 defines for us a new idea for encryption: design a system that is provably hard to break for a \textit{computationally bounded} adversary. Towards this end, they suggest a general scheme for many users A, B, C, etc. each having a pair of public and private keys, < \( K_p, K_s \)>.

The scheme is defined as usual by a triplet of PPT algorithms < \( G, E, D \) > (\( D \) is deterministic). Each user (we assume user A for convenience) initially uses \( G \) to generate his pair of keys \( G(1^k) = < P, S > \) for some security parameter \( k \). A then publishes his public key to a known directory available to all users and stores his secret key privately. To send a message to A, any other user looks up A’s public key in the directory and uses it with the encryption algorithm \( E \) to encrypt a message, \( c = E(m, P) \). To decrypt this message, A uses his secret key with the decryption algorithm \( D \), \( m = D(c, S) = D(E(m, P), S) \). This scheme straightforwardly generalizes to all users.

There are three properties to note about public key encryption. The first is that \( E \), \( D \), and \( P \) are all available publicly to any user or adversary, and \( E \) and \( D \) are easy to compute for known inputs. The second is that any user can send a message to any other user without having to meet in advance. Finally, the public directory is not a trusted intermediary in the sense that it cannot perform any kind of decryption, it only stores public keys.

As a quick aside, only \( G \) and \( E \) are PPT algorithms (\( E \) being PPT allows exponentially many cipher texts \( c \) to be produced from a single message \( m \)). If \( D \) were a PPT algorithm, we could possibly attempt “deniable computing,” in which even if forced, a user can “deny” the decryption of a particular cipher text to a particular message and an adversary will be unable to efficiently identify this deception. More details are omitted, however, as this subject is outside the realm of 6.875, and involves several open questions.

2 Trapdoor Function Model

Our first candidate for a public key encryption system is to use trapdoor functions. We are given a message space \( M \) with messages of length \( k \), and a collection of trapdoor functions \( F \), where for each function \( f_i \), we have trapdoor \( t_i \). We define \( (G, E, D) \) as follows:

- \( G \) generates a pair of keys consisting of a random member function of \( F \) and its corresponding trapdoor, \( G(1^k) = (P, S) = (f, t) \).
- \( E \) encrypts \( m \) by calculating \( c = E(P, m) = E(f, m) = f(m) \).
- \( D \) decrypts \( c \) by calculating \( m = D(S, c) = D(t, f(m)) = f^{-1}(f(m)) \).

3 Security

Is this elegant idea secure? We know from the hard-to-invert property of trapdoor functions that \( f(x) \) is impossible to efficiently invert for random \( x \). However, we encounter two problems. The first is that this statement only holds for \textit{random} \( x \). If restrict \( M \) to some special message space (i.e., \( \{0, 1\} \)), we may find \( f \) is easy to invert. Second, we know that, regardless of message space, a trapdoor function \( f \) may reveal partial information about \( x \) (for example, the least significant bit of the EXP function). We can generalize this to say that any deterministic \( E \) leaks partial information about \( x \).

In order to address this problem, we create a new definition of security:
Security: A (G, E, D) triplet is computationally secure if \( \forall \text{PPT} M, \forall \text{adv} A, \forall \text{poly} Q, \Pr[A(1^k, m_0, m_1, c) = 1|c \in E(m_0)] - \Pr[A(1^k, P, m_0, m_1, c) = 1|c = E(m_1)] < \frac{1}{Q(k)} \).

4 A Brief Preview of Probabilistic Public Key Encryption

In order to satisfy this new definition of security, we take advantage of the hardcore predicates of trapdoor functions. We do this by noting that a particular hardcore predicate is always found at a certain position for a given trapdoor function and given string length. We can take advantage of this by placing our message in exactly the hardcore predicate position of a longer string, and padding the rest with random bits. Thus to encrypt a message \( m \), we calculate \( c = E(m, i) = f_i(r_1mr_2) \) where \( r_1 \) and \( r_2 \) are generated at random, such that \( m \) is located in the hardcore predicate position of \( f \).