CSE 599b: Cryptography **(Winter 2006)**

Lecture 4: More Block Ciphers; Pseudorandom Generators

13 January 2006

Lecturer: Paul Beame *Scribe: Paul Beame*

0.1 Aside: Why two Feistel rounds are not enough

Suppose that $f : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and we have a block cipher with block-length $2n$ and key length $2k$ and key defined for $K = K_1K_2$ by $M = (L, R) \to C = (L'', R'')$ using two Feistel rounds:

$$\begin{aligned} L' &\leftarrow R \\ R' &\leftarrow L \oplus f_{K_1}(R) \\ L'' &\leftarrow R' \\ R'' &\leftarrow L' \oplus f_{K_2}(R') \end{aligned}$$

Then for two messages M_1 and M_2 with the same right half say $M_1 = L_1R$, $M_2 = L_2R$, observe that the left halves L_1'' and L_2'' of their corresponding encryptions C_1 and C_2 are $L_1'' = L_1 \oplus f_{K_1}(R)$ and $L_2'' = L_2 \oplus f_{K_1}(R)$. Thus $L_1'' \oplus L_2'' = L_1 \oplus L_2$ which is essentially as bad as using a one-time pad on the left half of the input.

1 On the security of DES

Given a single (M, C) pair with $C = DES_K(M)$ and the fact that DES has a key length of 56 bits and brute force key search will succeed on average in 2^{55} trials to find a K' such that $DES_{K'}(M) = C$; moreover it is unlikely that even two different keys will work for the single (M, C) pair since there are more ciphertexts than keys so almost surely $K' = K$.

More sophisticated attacks have been developed, although neither is particularly practical for DES:

Differential cryptanalysis [Biham, Shamir 89] Examines how the differences between closely related messages get propagated through the rounds of the block cipher. Can break DES after seeing 2^{47} ciphertexts of chosen plaintexts. This attack was known to the NSA, but not made public, during the development of DES. It has been observed that the design of the S-boxes in DES makes it clear that they were aware of it since even small changes to their design would make DES much more vulnerable to this attack. Differential cryptanalysis was fatal for some designs that had been proposed as alternatives to DES and has been effective recently in breaking current cryptographic hash function designs.

Linear cryptanalysis [Matsui 94] Looks for correlations between parts of DES and linear functions. Can break DES after only 2^44 ciphertexts of known plaintexts but requires massive storage to compare them.

To get an idea of the numbers, on current machines there are roughly 2^{30} clock cycles in a second and roughly 2^{30} seconds in 30 years. This should already give one pause in using DES, especially given the fact that the brute force attacks parallelize trivially. Nearly a decade ago, after it had been suggested that one could build a machine to break DES very easily, the Electronic Freedom Foundation did just that and was able to break DES in an average of 56 hours. This led to attempts to strengthen DES.

2 Double DES, Triple DES, DESX

All of these are 64 bit ciphers that are extensions of DES but use much larger keys.

Double DES

Here $2DES(K_1K_2, M) = DES_{K_1}(DES_{K_2}(M))$. This yields a key length of 112 bits and is twice as slow as DES. Ideally one would like this to have brute force-style security that would require 2^{111} trials. However, a very simle "meet in the middle attack" can find the key in only 2^{57} trials (although this requires a lot of storage and is not practical). The idea is the following: Given an (M, C) pair compute all $DES_S(M)$ for varying keys S and $DES_T^{-1}(C)$ for varying keys T and use a large hash table to detect collisions between these two sets. If $A = DES_S(M) = DES_T^{-1}(C)$ is a collision then $DES_T(DES_S(M)) = C$ and TS is a good candidate key.

Triple DES

There are two versions both with 168 bit keys and each is three times as slow as DES:

$$\begin{aligned} 3DES3(K_1K_2K_3, M) &= DES_{K_3}(DES_{K_2}^{-1}(DES_{K_1}(M))) \\ 3DES2(K_1K_2, M) &= DES_{K_2}(DES_{K_1}^{-1}(DES_{K_2}(M))) \end{aligned}$$

Each of these uses DES^{-1} (which is essentially the same difficulty as DES) in the middle in order to achieve backward compatibility since

$$3DES3(KKK, M) = 3DES2(KK, M) = DES_K(DES_K^{-1}(DES_K(M))) = DES_K(M).$$

The two versions of Triple DES so far do not appear to have particular vulnerabilities but they are both particularly slow in software and not great in hardware so alternatives were sought that would be faster.

DESX

DESX also has 184-bit keys but is essentially as fast as DES:

$$DESX(KK_1K_2, M) = K_2 \oplus DES(K, K_1 \oplus M)$$

Moreover, backward compatibility comes from setting $K_1 = K_2 = 0^{64}$. It is a bit unclear how secure DESX is but there was also concern about the inefficiency of its 64 bit blocksize.

3 AES/Rijndael

In the mid 1990's when it was clear that DES needed to be completely replaced there was an open competition to design a replacement which would be a 128-bit block cipher.

The winning design which became the Advanced Encryption Standard or AES was Rijndael, which is an amalgam of the two authors' names. Unlike all the others it is permutation that is not derived from the Fiestel method (and seems to use its bits a little more efficiently since Feistel ciphers only mix half the bits as a time). There are 3 versions depending on the key length. AES128, AES192, and AES256. There are all pretty similar so we'll consider AES128.

AES128 expands the 128 bit key K into 11 round keys each of 128 bits using similar ideas to DES. There are 10 rounds and at the start and end and between each pair of rounds a round key is XOR-ed with the current 128-bit scrambling of the message. The main difference is what goes on in each round.

The main thing they use is arithmetic in $GF(2^8)$, the finite (Galois) field of $2^8 = 256$ elements (conveniently one byte per element). Elements of $GF(2^8)$ are described by polynomials that are taken modulo 2 and modulo an 'irreducible' polynomial $m(x)$ of degree 8. In particular, the polynomial used is $m(x) = x^8 + x^4 + x^3 + 1$ (which the authors say just happened to be the first one in a standard list of such polynomials). Thus the byte $b_7 \dots b_1 b_0$ represents the polynomial $b_7 x^7 + \cdots + b_1 x + b_0$ in $GF(2^8)$ which has been reduced mod $m(x)$ and 2.

To add two polynomials in this representation of $GF(2^8)$ one simply adds their coefficients and reduces the result mod 2. To multiply, one multiplies them as polynomials (which is just a shift and add mod 2) and uses the identity that $x^8 + x^4 + x^3 + 1 = 0$ which implies that $x^8 = x^4 + x^3 + 1$ (since additing and subtracting are the same mod 2) and thus one can do shift and mod 2 additions also to get rid of any coefficients of $x^{14}, x^{13}, ..., x^8$.

For a field one also needs multiplicative inverses. This is why $m(x)$ had to be irreducible (doesn't factor mod 2). This is the analog of being a prime over the integers. Euclid's algorithm shows that if $gcd(a, m) = 1$ then one can solve $az \mod m = 1$ for z; moreover if m is prime and $a \not\equiv 0 \mod m$ then such a z is unique mod m and we write $x = a^{-1} \mod m$. In the case of $GF(2^8)$, $m(x)$ is irreducible so for any polynomial $a(x) \in GF(2^8)$, with $a(x) \neq 0$, the $gcd(a(x), m(x)) = 1$ and thus we can find a *inverse* polynomial $INV(a)(x) = (a(x))^{-1} \mod m(x)$ such that $a(x) \cdot INV(a)(x) \mod m(x) = 1$. Moreover, we can extend this map that takes a to $INV(a)$ to a permutation of all of $GF(2^8)$ by defining $INV(0) = 0$. A simple 256 byte look-up table can be used to compute such inverses.

Now we can describe the internals of a round between the XOR-ing in of the round keys. The inversion operation above is used at the start of each round on each byte separately as the analog of the S-boxes of DES. For the remaining two operations one thinks of the 16 bytes as being arranged in a 4×4 square. The remaining two operations are a column scrambling operation done separately one each column followed by a row shifting. The latter simply cyclically shifts the second row by 1, the third row by 2 and the last row by 3. The more interesting part is the column scrambling operation. In this one views each column as a degree 3 polynomial in $GF(2^8)[y]$. That is a polynomial in y whose coefficients are polynomials in x of degree at most 7. Each column is multiplied by a different degree 3 polynomial in $GF(2^8)[y]$ and the result is computed mod the polynomial $y^4 + 1$ to derive a new degree 3 polynomial in $GF(2^8)[y]$. That is, multiplication is done modulo $y^4 + 1$, modulo $x^8 + x^4 + x^3 + 1$ and modulo 2. Note that the polynomial $y^4 + 1$ is NOT irreducible but the multipliers done have a common factor with $y^4 + 1$ over $GF(2^8)$ and therefore this operation can be inverted. Performing this operation is also easy.

AES128 is much faster in software than Triple DES or its other competitors. It has the advantage of having its design decisions public and had extensive public testing before it was adopted (often by people who had vested interests in showing that their competing cipher was better). For more details of the design decisions for AES see: http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf

4 Towards Security of Block Ciphers

Part of our intuitive notion of being secure required that the encryptions of messages look random and distinct. However, if we have $E_K(M_1), E_K(M_2), ...$, once we have more bits in some fixed messages $M_1, M_2, ...$ than we have bits in K we will essentially have more random-looking bits of output than we have bits of random key. Thus a block cipher will have to create pseudorandom bits. Therefore, before we can formalize security of block ciphers we will need to formalize pseudorandomness.

5 Formal notions of security and pseudorandomness

There are two natural mathematically sound ways one can go about defining security and pseudorandomness that exemplified by the notes by Bellare and Goldwasser and Bellare and Rogaway, respectively. Each has its advantages and disadvantages.

Asymptotic security Developed by Blum, Goldwasser, Micali, and Yao in the 1980's.

- Allow all probabilistic polynomial time algorithms and adversaries
- Security=negligible probability of failure (the adversary wins)

The positive aspects are that it is good to be able to say that systems are secure against all reasonable adversaries but one must discuss asymptotics and therefore one must have an infinite family of systems and the 'security' of, say, a 128-bit system is not meaningful.

Concrete security Developed by Bellare, Kilian, and Rogaway in the early 1990's as a refinement of asymptotic security.

- Allow probabilistic algorithms with specific bounds on their running time t and number of queries q.
- Bound failure probability as an exact function of t, q, key-size k, message length n, etc.

The positive aspects are that one can talk about how secure systems of a specific size are (without an infinite family even existing) and so one can measure the security of practical systems, although one can never say that a system is secure. The main negative aspect for us is that using concrete security often means that one simply has to spend more time analyzing exactly the same arguments as in the asymptotic complexity case but there are more painful details to handle. We will use the asymptotic complexity formulation since it will have fewer of those details to deal with.

6 Pseudorandom Number Generators (PRNG)

Recall

Definition 6.1. *An* ensemble $\mathcal{E}$ *is an infinite family* $\{\mathcal{E}_n\}_n$ *of distributions where* $\mathcal{E}_n$ *is a distribution on* $\{0,1\}^n$.

Definition 6.2. *An function* $\nu : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ *is* negligible *if and only if* $\nu(n)$ *is* $1/n^{\omega(1)}$*; i.e.* $\nu(n)$ *goes to 0 fast than 1 over any polynomial in* n.

Definition 6.3. *A* polynomial-time statistical test *is a probabilistic polynomial time algorithm* A.

We will write PPT for 'probabilistic polynomial-time algorithm'.

Definition 6.4. *Two ensembles* $\mathcal{E}$ *and* $\mathcal{D}$ *are* computationally (polynomially) indistinguishable *if and only if for all PPT* A*, the function* ϵ *given by*

$$\epsilon(n) = \Pr[A(x) = 1 \mid x \leftarrow \mathcal{E}_n] - \Pr[A(x) = 1 \mid x \leftarrow \mathcal{D}_n]$$

is negligible. (We think of $\epsilon(n)$ *as the advantage that* A *has in discriminating* $\mathcal{E}$ *from* $\mathcal{D}$ *and therefore sometimes write* $\epsilon(n) = Adv_A^{\mathcal{E},\mathcal{D}}(n)$*. Also we sometimes express* $\epsilon(n)$ *simply as* $\Pr[A(\mathcal{E}_n) = 1] - \Pr[A(\mathcal{D}_n) = 1]$.

Uniform Distribution We let $\mathcal{U}$ be the uniform ensemble where $\mathcal{U}_n$ is the uniform distribution on $\{0,1\}^n$.

Definition 6.5. *An ensemble* $\mathcal{E}$ *is* pseudorandom *if and only if it is computationally indistinguishable from* $\mathcal{U}$.

This gives the intuitive basis for the definition of pseudorandom generators.

Definition 6.6. *A deterministic polynomial time computable function $G : \{0,1\}^* \to \{0,1\}^*$ is a* pseudorandom generator (PRNG) *if and only if*

- *Length Increasing: $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ where $\ell(n) > n$.*
- *Pseudorandom: For every PPT A, ϵ is negligible where $\epsilon(n) = \Pr[A(y) = 1 \mid x \leftarrow \mathcal{U}_n; y \leftarrow G(x)] - \Pr[A(y) = 1 \mid y \leftarrow \mathcal{U}_{\ell(n)}]$.*

Intuitively, G being pseudorandom requires that the ensembles $\mathcal{U}$ and $G(\mathcal{U})$ are computationally indistinguishable, i.e. $G(\mathcal{U})$ is pseudorandom, but there is a slightly annoying aspect of the getting the lengths to match so we spell things out in detail.

After seeing that a second use was enough to kill the security of one-time pads or a simple information-theoretic definition of MACs one might be concerned that maybe we are giving the adversary A too little access to G. Maybe some larger number of queries would allow A to do better? To discuss this we need another concept.

Definition 6.7. *An ensemble $\mathcal{D}$ is* polynomial-time samplable *if and only if there is a PPT S such that $S(1^n)$ produces elements of $\{0,1\}^n$ distributed as $\mathcal{D}_n$.*

Clearly $\mathcal{U}$ is polynomial-time samplable and (not worrying too much about indices) $G(\mathcal{U})$ is polynomial-time samplable. We will see next time that for two computationally indistinguishable ensembles that are polynomial-time samplable, if we allow PPT algorithms access to a polynomial number of queries rather than just one query then they still have negligible advantage. This will introduce the basic idea of the reduction method we will use throughout the course and will introduce the 'Hybrid Argument' that appears frequently in analyzing security.