ECS 227: Modern Cryptography
Lecture 3

Reducibility and the Modeling of Block Ciphers

Phillip Rogaway
16 January 1996

3 A First Reduction

Last time we sketched the *Secure Hash Algorithm,* SHA. Remember how this worked? There was a *compression function* sha : $\{0,1\}^{160} \times \{0,1\}^{512} \to \{0,1\}^{512}$ which was extended to a map SHA : $\{0,1\}^{<2^{64}} \to \{0,1\}^{160}$ by the following recipe: to compute SHA(x), where $|x| = \ell < 2^{64}$,

> Append to x a `1`; then append the minimum number of `0`s such that the resulting string will have length which is 64 less than a multiple of 512; then append ℓ, encoded as a 64-bit binary number. Let the resulting string M consist of n 512-bit blocks, $M = M_1 M_2 \cdots M_n$.
>
> Let $k =$ `0x6745`$\cdots$`E1F0` be the constant specified in the SHS.
>
> **for** $i \leftarrow 1$ **to** n **do**
>
> $k \leftarrow \text{sha}(k, M_i)$.
>
> **return** k

3.1 Definitions are tricky

In today's lecture we want to show that SHA is "good" as long as sha is "good." But what does it mean to say that SHA is "good"? What does it mean to say that sha is "good"? (This is a different question).

The obvious place to look for these answers is in the SHS itself. There we find:

> The SHA is designed to have the following properties: [*Condition A*] it is computationally infeasible to find a message which corresponds

> to [*ie., hashes to*] a given message digest, or [*Condition B*] to find two different message which produce the same message digest.

(Italics my own.) Now, what do you think of the above statement? (Don't worry about the exact meaning of "computationally infeasible"; for now, this isn't my concern.)

Condition A is very hard to make sense of: what does it mean a *given* message digest? Suppose I choose a string x and compute $k = \mathrm{SHA}(x)$. Then isn't k a *given* message digest for which it is computationally *trivial* to find a message (ie., x) which hashes to it? The above remark isn't meant to be a flippant: in fact, this problem seems very hard to sidestep— it is really unclear how we can exclude in a definition an adversary's ability to construct messages in exactly this sort of way. One might put forward that the authors of the SHS *really* meant that it should be computationally infeasible to find a preimage for a *random* message digest, say. That would make sense, at least, but it's pretty far from what was said.

Condition B might seem clearer, but it too is not without difficulties. When we say it is computationally infeasible to find two different messages, we're somehow asserting the inexistence of a computationally reasonable algorithm. That is, we might try to formalize Condition B by saying something like

> [**Bogus Definition**] There is no computationally efficient algorithm that, on input of the description of SHA, finds a pair of distinct strings M and M' such that $\mathrm{SHA}(M) = \mathrm{SHA}(M')$.

But this makes no sense because *of course* there is a computationally efficient algorithm that finds such an M and M': the particular algorithm which *knows* such an M and M' and immediately outputs them.

The above definitional difficulty can be overcome, but only by changing ones view of what an SHA-like primitive *is* (it can not be some one, fixed map). You will pursue this idea in homework, but, in the mean time, we are going to manage to give a rigorous theorem even in the *absence* of a proper definition of our goal.

For now let us think, informally, of Condition B as being our goal for an SHA-like primitive. We'll choose an exactly analogous goal for it compression function:

> **Informal Definition**— A hash function H is collision intractable if no one can find distinct M, M' such that $H(M) = H(M')$.
>
> **Informal Definition**— A compression function h is collision intractable if no one can find distinct (k, x), (k', x') such that $h(k, x) = h(k', x')$.

We're going to show that H is collision intractable if h is collision intractable. In other words, if you could find a collision in H then you could also find a collision in h. Thus it is enough, if you want to construct a collision-intractable hash function, to construct a collision-intractable compression function.

Observe in what follows how we neatly manage to give a theorem which captures the reduction *without* our ever having formally defined what our goal. This is not atypical; it is often possible to get a formal theorem (which says exactly the "right" thing) even without a definition of the intent of our primitives. In some sense, the intent of the primitives is implicit in the statement of the reduction.

3.2 Collision intractability of iterated hash functions

Let us fix some general notation: for some parameters k, n and b, $k < n < b$, we have a hash function $H : \{0,1\}^{<2^k} \to \{0,1\}^n$ defined from a compression function $h : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^b$ using the same method as we used to define SHA from sha. We show that there's a fixed, efficient algorithm that finds a collision in h given a collision for H.

Theorem 3.2.1 *Then there is an (explicitly given) linear-time algorithm A for which the following is true. Let H be an iterated hash function, defined in terms of compression function $h : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^b$. Provide A with an $O(b)$-time oracle which computes h. Suppose $M \neq M'$ are such that $H(M) = H(M')$. Let A, on input M, M', finds a $(k,x) \neq (k',x')$ such that $h(k,x) \neq h(k',x')$.*

Proof. Let $Mp = M_1 \cdots M_n$ be the padded version of M, partitioned into b-bit blocks. Let $M'p' = M'_1 \cdots M'_{n'}$ be the padded version of M', partitioned into b-bit blocks. We consider three cases:

Case 1 $|M| = |M'|$.

Case 2 $|M| \neq |M'|$ but $n = n'$, and

Case 3 $|M| \neq |M'|$ but $n \neq n'$.

In class we only did Case 1, which follows. Case 2 is essentially identical and Case 3 is left for a homework.

We now describe algorithm A. It hashes M and M' using H. In the process, A computes

$$k_2 = h(k_1, M_1), k_3 = h(k_2, M_2), \ldots, k_n = h(k_{n-1}, M_{n-1}), k_{n+1} = h(k_n, M_n)$$

($k_{n+1} = H(M)$), and

$$k'_2 = h(k'_1, M'_1), k'_3 = h(k'_2, M'_2), \ldots, k'_n = h(k'_{n-1}, M'_{n-1}), k'_{n+1} = h(k'_n, M'_n).$$

($k'_{n+1} = H(M')$, and $k_1 = k'_1$.) Algorithm A outputs the first $1 \leq i \leq n$ such that $h(k_i, M_i) = h(k'_i, M'_i)$ but $(k_i, M_i) \neq (k'_i, M'_i)$. We now must argue that such an i necessarily exists.

By assumption, $H(M) = k_{n+1} = k'_{n+1} = H(M')$. So, if $(k_n, M_n) \neq (k'_n, M'_n)$, then we are done. Otherwise —that is, $M_n = M'_n$, $k_n = k'_n$—

we consider whether or not $(k_{n-1}, M_{n-1}) \neq (k'_{n-1}, M'_{n-1})$. If these are unequal, then, again, we are done. Otherwise (they are equal) and we consider if $(k_{n-2}, M_{n-2}) \neq (k'_{n-2}, M'_{n-2})$. Continuing in this way, we either find a (largest) i such that $(k_i, M_i) \neq (k'_i, M'_i)$ but $h(k_i, M_i) = h(k'_i, M'_i)$, or we establish that $M_1 = M'_1$, $M_2 = M'_2$, …, $M_n = M'_n$. The later is impossible by $M \neq M'$. $\diamondsuit$

4 Modeling Block Ciphers

We now turn our attention to the modeling of block ciphers.

4.1 Syntax

With DES as our example we've already examined the "syntax" of a block cipher.

Definition 4.1.1 *A* Finite Pseudorandom Permutation *(Finite PRP), or* block cipher*, with* key length κ *and* block length ℓ *is a map*

$$F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$$

where, for every $a \in \{0,1\}^\kappa$*,* $F(a, \cdot)$ *is a permutation on* $\{0,1\}^\ell$*.*

We'll denote $F(a, x)$ by $F_a(x)$.

We've seen how the Feistel-construction guarantees that DES will be a block cipher (that is, it guarantees that $\mathrm{DES}_a(\cdot)$ is a permutation).

Again, the above describes only the *syntax* of a block cipher— it says nothing about the security. For example, under the definition above, the identity cipher (ie, the block cipher which, for every key a, is the identity function on $\{0,1\}^\ell$) is a block cipher, though certainly it is not a good one! Now we want to *quantify* the "goodness" of a block cipher.

4.2 Security of a block cipher

The intuition is that a "good" block cipher should "look like" a random family of permutations, in the following sense. Suppose we give an adversary D one of two types of boxes, as described below. Which type of box we give her is determined by flipping a fair coin.

- We might give the adversary a block cipher box. This box is defined by choosing a random key $a \in \{0,1\}^\kappa$ and giving the adversary a box which computes $F_a(\cdot) : \{0,1\}^\ell \to \{0,1\}^\ell$. We emphasize that when you run the adversary with the block cipher box the key a is only chosen once: each query $x_1, x_2, \ldots$ is answered with the same key: $F_a(x_1), F_a(x_2), \ldots$.

- Alternatively, we might give the adversary a random permutation box. This is determined by selecting, at random, one of the $2^\ell!$ possible permutations on $\{0,1\}^\ell$. Again, you make the choice only once and give this (one) box to the adversary.

A block cipher box and a random permutation box look identical to the adversary. They're the same weight, size, shape, color, etc.. The adversary's *only* way to distinguish these two types of boxes is to experiment with them, examining their input/output behavior.

The adversary's job is to guess which type of box she has. Some adversaries might do very well distinguishing a block cipher box from a random permutation box: they get it right almost every time. Some adversaries do a lousy job: their prediction as to the type of box they have is no better than a naive guess. We want to quantify how well an adversary does in this game. We can do this in a couple ways.

Let $\mathcal{P}_\ell$ be the set of permutations on ℓ-bit strings.

Definition 4.2.1 *Let D be a Finite PRP adversary — an adversary with an oracle that outputs a bit. Let F be a block cipher with key length κ and block length ℓ. We define D's* advantage *at distinguishing F from a random permutation by*

$$\mathrm{Adv}_D^{F,\pi} = \Pr[a \leftarrow \{0,1\}^\kappa :\ D^{F_a(\cdot)} = 1] - \Pr[\pi \leftarrow \mathcal{P}_\ell :\ D^{\pi(\cdot)} = 1]$$

This definition above didn't exactly mimic the game we described: a closer, but somewhat more complicated-looking translation, would have been the following:

Definition 4.2.2 *Let D be a Finite PRP adversary and let F be a block cipher with key length κ and block length ℓ. Define*

$$\begin{aligned}\overline{\mathrm{Adv}}_D^{F,\pi} &= 2\cdot\Pr[b \leftarrow \{0,1\}; \text{if } b=0 \text{ then } g_b \leftarrow \mathcal{P}_\ell \\ &\qquad\qquad \text{else } \{a \leftarrow \{0,1\}^\kappa;\ g_b \leftarrow F_a\} :\ D^{g_b(\cdot)} = b] - 1.\end{aligned}$$

The multiplication by 2 and subtraction by 1 are scaling factors so that "perfect advantage" has a numerical value of 1; being "absolutely wrong" has a numeric value of -1; and having "no advantage" —solving the game correctly exactly as often as luck would have it— corresponds to advantage of 0. These same values are used in the earlier notion of advantage. By employing this scaling factor it is easy to see that the two definitions above are equivalent:

Proposition 4.2.1 *Let D be a Finite PRP adversary and let F be a block cipher. Then $\mathrm{Adv}_D^{F,\pi} = \overline{\mathrm{Adv}}_D^{F,\pi}$.*

Proof. The proof is by simple manipulation of the probabilities:

$$\overline{\mathrm{Adv}}_D^{F,\pi} = 2\Pr[D^{g_b(\cdot)} = b] - 1$$

$$
\begin{aligned}
&= 2(\Pr[D^{g_b(\cdot)} = b | b = 0] \Pr[b = 0] + \Pr[D^{g_b(\cdot)} = b | b = 1] \Pr[b = 1]) - 1 \\
&= 2(\Pr[D^{\pi(\cdot)} = 0](1/2) + \Pr[D^{f_a(\cdot)} = 1](1/2)) - 1 \\
&= \Pr[D^{\pi(\cdot)} = 0] + \Pr[D^{f_a(\cdot)} = 1] - 1 \\
&= (1 - \Pr[D^{\pi(\cdot)} = 1]) + \Pr[D^{f_a(\cdot)} = 1] - 1 \\
&= \Pr[D^{f_a(\cdot)} = 1] - \Pr[D^{\pi(\cdot)} = 1]) \\
&= \mathrm{Adv}_D^{F,\pi} \qquad \diamondsuit
\end{aligned}
$$

Now, with a notion of advantage, we can define our notion of security of a block cipher. This is not an "absolute" notion: security is a question of how much advantage an adversary can gain as a function of the amount of resources which she spends

Definition 4.2.3 *Let $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a Finite PRP. Adversary D is said to (t, q, ϵ)*-distinguish F from a random permutation *if D runs in at most t computational steps, makes at most q queries of her oracle, and $\mathrm{Adv}_D^{f,\pi} \geq \epsilon$.*

Above, the "size" of D can not really be ignored, and the simplest fix is to include the size of D as part of t. (That is, t is understood to be the computational steps of D plus plus the length of a description of D.) Another approach is to think of D as a family of circuits over two-input AND, OR, and NOT gates, as well as gates which serve as the oracle. We will return to this topic in the next lecture.

A "good" block cipher doesn't allow *any* efficient adversary to achieve a reasonable advantage. In other words, if $\mathrm{Adv}_D^{F,\pi}$ is big enough to concern you, then either t or q should be big enough that D is not, in fact, practical enough to cause you to worry.

Definition 4.2.4 *A Finite PRP $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$ is said to be (t, q, ϵ)*-secure *if every adversary D which runs in time $\leq t$ and makes $\leq q$ oracle queries has $\mathrm{Adv}_D^{f,\pi} \leq \epsilon$.*