Last Revised: 03/02/96.

The presentation in these notes was motivated by the notes of Rafial Ostrovsky

ECS 227: Modern Cryptography
Lectures 11, 12

The Goldreich/Levin Hard-Core Bit

Phillip Rogaway
{15, 20} *February 1996*

8.7 Encrypting with an arbitrary trapdoor permutation

We've claimed (without proof) that a secure encryption of the bit b under RSA is to select a random $r \in \mathsf{Z}_N^*$ and then take $\mathcal{E}(b) = \langle r^e \bmod N,\ \mathrm{lsb}(r) \bmod b\rangle$ as the encryption of b. Instead of proving the correctness of this scheme, we're going to prove something more general. Here's the strategy.

Start with an arbitrary trapdoor permutation, $\mathcal{F}_1$. We will make a "padded version" of $\mathcal{F}_1$, which we call $\mathcal{F}_2$. Whereas $\mathcal{F}(1^k)$ returns (f_1, f_1^{-1}), the generator $\mathcal{F}_2$ first computes an (f_1, f_1^{-1}), but then returns (f_2, f_2^{-1}), where

$$f_2(x \,.\, p) \quad = \quad (f_1(x) \,.\, p)$$

for $|p| = |x| = n$. It's not important how we define f_2 on strings of odd length.

Now we define

$$B(x \,.\, p) \quad = \quad x \odot p$$

as the mod-2 inner product of x and p. For example,

$$10110 \odot 00111 = 1 \cdot 0 \oplus 0 \cdot 0 \oplus 1 \cdot 1 \oplus 1 \cdot 1 \oplus 0 \cdot 1 = 0.$$

Our new encryption scheme is as follows:

> Let $\mathcal{F}$ be a trapdoor permutation generator. When the security parameter is k, a party selects $(f, f^{-1}) \leftarrow \mathcal{F}(1^k)$ and uses f for his public key, and f^{-1} for his secret key. To encrypt a bit b under a

public key f, choose a random $r \in \{0,1\}^k$, and a random $p \in \{0,1\}^k$. The encryption of b (using r and p) is $\mathcal{E}(b) = \langle f(r),\ p,\ (r \odot p) \oplus b\rangle$. To encrypt a message $b_1 \cdots b_n$, separately encrypt each of its bits.

Decryption proceeds in the obvious way.

To show that this scheme "works" we will need to show that $B(x\ .\ p)$ is hard to guess if $\mathcal{F}$ is a trapdoor permutation. None of this has anything to do with the trapdoor-ness of $\mathcal{F}$, so we'll linguistically simplify and address the problem of making a hard predicate B for a one-way function f.

8.8 Notion of a hard-core bit

We have the following definition for an adversary's advantage in guessing a bit B about the preimage of a one-way function f.

Definition 8.8.1 *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a function, let $B : \{0,1\}^* \to \{0,1\}$ be a predicate, and let A be an adversary. We say that A (t,ϵ)-guesses B on 1^k if*

$$\begin{aligned} \mathrm{Adv}_A &= \Pr\Big[x \leftarrow \{0,1\}^k;\ y \leftarrow f(x):\ A(y) = B(x)\Big] - 1/2 \\ &\geq \epsilon \end{aligned}$$

and A runs in time at most t.

Usually we've been scaling this up, to put the advantage in the range $[0,1]$. instead of $[-0.5, 0.5]$. This time, we won't bother.

For an asymptotic definition we define B to be a hard-core bit for f if, for any adversary A which $(t(k),\epsilon(k))$-guesses B on 1^k, $\epsilon(k)$ is negligible as long as $t(k)$ is polynomial.

Informally, B is hard-core for $\mathcal{F}$ if no adversary, given $y = f(x)$, can guess $B(x)$ and be right significantly better than half of the time. This is exactly what we want to encrypt a bit.

8.9 The Goldreich-Levin hard-core bit

Let us now return to the construction of $\mathcal{F}_2$ from $\mathcal{F}_1$, and $B(x\ .\ p) = x \odot p$. We have the following lovely result of Goldreich and Levin. As explained above, we state this theorem for one-way functions instead of trapdoor permutations, the trapdoorness being irrelevant to the construction. We state the theorem in the asymptotic sense, but the exact bound will be apparent from our proof.

Theorem 8.9.1 *Let f_1 be a one-way function, and let $f_2(x\ .\ p) = (f_1(x), p)$. Let $B(x\ .\ p) = x \odot p$. Then B is a hard-core bit for f_2*

The proof occupied a good deal of time in lecture, but it was quite interesting. I'll make the lecture notes briefer than was the lecture, omitting a good deal of the intuition and development which we described.

To prove this theorem, we start off with an adversary A_B which, given $y = f_1(x)$ and p, does a good job of guessing $x \odot p$:

> Given— **Adversary** A_B**:** On input $y \,.\, p$, where $y = f_1(x) \in \{0,1\}^k$, adversary A_B guesses $x \odot p$. Adversary A_B is right a fraction $1/2+\epsilon$ of the time, where $\epsilon(k) \geq 1/\mathrm{poly}(k)$, for some particular polynomial, on some infinite set of indices K. To be clear, what we know about A_B is:
>
> $$\begin{aligned} \mathrm{Adv}_{A_B} &= \Pr\Big[x, p \stackrel{R}{\leftarrow} \{0,1\}^k;\ y \leftarrow f_1(x) : \\ &\qquad A_B(y \,.\, p) = x \odot p\Big] - 1/2 \\ &\geq \epsilon\ . \end{aligned}$$
>
> For notational simplicity, assume that f_1 maps strings of k bits to strings of k bits.

Somehow we need to construct from A_B an adversary A_{f_1} which actually inverts f_1. Think about that for a moment and appreciate the difficulty of this goal! Adversary A_{f_1} should manage to find a preimage for $y = f(x)$ some inverse-polynomial fraction of the time, on some infinite index set:

> Construct— **Adversary** A_{f_1}**:** On input y, find $x = f^{-1}(y)$. We need that
>
> $$\begin{aligned} \mathrm{Adv}_{A_{f_1}} &= \Pr\Big[x \leftarrow \{0,1\}^k;\ y \leftarrow f_1(x) : \\ &\qquad f_1(A_{f_1}(y)) = y] \\ &\geq \epsilon' \end{aligned}$$
>
> for some e' which is at least an inverse polynomial, on some infinite index set K.

If we start from a trapdoor permutation instead of a one way function, what we need to do changes a little. We state what's different, then continue our explanation with the prior notions.

> Given— **Adversary** A_B**:** On input $1^k, f, y, p$, where $y = f(x)$, guess $x \odot p$:
>
> $$\begin{aligned} \mathrm{Adv}_{A_B} &= \Pr\Big[(f, f^{-1}) \leftarrow \mathcal{F}(1^k);\ x, p \stackrel{R}{\leftarrow} \{0,1\}^k;\ y \leftarrow f(x) : \\ &\qquad A_B(1^k, f, y \,.\, p) = x \odot p\Big] - 1/2 \\ &\geq \epsilon \end{aligned}$$

Construct—

Adversary A_{f_1}**:** On input $1^k, f, y$, find $x = f^{-1}(y)$:

$$\begin{aligned}
\mathrm{Adv}_{A_{f_1}} &= \Pr\Big[(f, f^{-1}) \leftarrow \mathcal{F}(1^k);\ x \leftarrow \{0,1\}^k;\ y \leftarrow f(x) : \\
&\qquad A_{f_1}(1^k, f, y) = x\Big] \\
&\geq \epsilon' .
\end{aligned}$$

Now we're going to be describe algorithm A_{f_1}, which calls algorithm A_B as a subroutine. In class we developed this algorithm in stages. For the notes, we'll give the final algorithm, then explain the analysis. The notation $p^{(j)}$ denotes the string p with bit number j flipped, where bits are numbered least significant to most significant, beginning at 1. For example, if $p = 1011101$, then $p^{(3)} = 1011\underline{0}01$.

Algorithm $A_{f_1}(y)$ — *// Try to invert f_1 at y.*

Let $k \leftarrow |y|$.

Let $m =$ *SOMETHING.* — *// Value given in analysis*
Let $t = \lceil \lg(m+1) \rceil$.

Choose $q_{t-1}, q_{t-2}, \ldots, q_1, q_0 \stackrel{R}{\leftarrow} \{0,1\}^k$.
Define $p_1, \cdots, p_m$ by: $p_1 \leftarrow q_0$; $p_2 \leftarrow q_1$; $p_3 \leftarrow q_0 \oplus q_1$; $p_4 \leftarrow q_2$;
$p_5 \leftarrow q_2 \oplus q_0$; $p_6 \leftarrow q_2 \oplus q_1$; $p_7 \leftarrow q_2 \oplus q_1 \oplus q_0$; $p_8 \leftarrow q_3$; etc..

Choose $\beta_{t-1} \cdots \beta_1 \beta_0 \stackrel{R}{\leftarrow} \{0,1\}^t$ — *// Guess values for* $\beta_\ell \stackrel{\text{hope}}{=} x \odot q_\ell$
// Assuming this guess correct, compute values for each $b_i = x \odot p_i$.
for $i \leftarrow 1$ **to** m **do** $b_1 \leftarrow \beta_0$; $b_2 \leftarrow \beta_1$; $b_3 \leftarrow \beta_0 \oplus \beta_1$; $b_4 \leftarrow \beta_2$;
$b_5 \leftarrow \beta_2 \oplus \beta_0$; $b_6 \leftarrow \beta_2 \oplus \beta_1$; $b_7 \leftarrow \beta_2 \oplus \beta_1 \oplus \beta_0$; $b_8 \leftarrow \beta_3$; etc..

for $j \leftarrow 1$ **to** k **do**
  // Make a good guess for $x[j]$.
  for $i \leftarrow 1$ **to** m **do**
    Compute $guess_i^j \leftarrow b_i \oplus A_B(y \,.\, p_i^{(j)})$.

  Set $x[j] = 1$ if $\sum_{i=1}^m guess_i^j \geq m/2$; 0 otherwise.

if $f(x) = y$ **then return** x **else** algorithm failed.

Pairwise Independence. While $p_1, \ldots, p_m$ are not independent, are still *pairwise independent.* This means that, for all $i \neq j$, $\Pr[\mathsf{P}_i = p_i \wedge \mathsf{P}_j = p_j] =$

$\Pr[\mathsf{P}_i = p_i] \cdot \Pr[\mathsf{P}_j = p_j]$. Beyond that, each p_i is uniformly distributed. These two properties are given by the following proposition:

Proposition 8.9.1 Let $\mathsf{P}_1, \cdots, \mathsf{P}_m$ be the random variables corresponding to $p_1, \cdots, p_m$ in algorithm A_{f_1}. Then, for all $1 \leq i < j \leq m$, the random variable $\mathsf{P}_i \parallel \mathsf{P}_j$ is uniformly distributed in $\{0,1\}^{2k}$.

To see the proposition above, note that there is some q_s which is included in the sum which defines p_i, but which is not included in the sum which defines p_j. Similarly, there is some q_t which is included in the sum which defines p_j, but which is not included in the sum which defines p_i. The values of s and t depend only on i and j. So P_i and P_j consist of (possibly correlated) stuff, xor'ed, respectively, with two uncorrelated, uniformly distributed random variables.

Good x-values. What would seem to be a major difficulty in the analysis is that the guarantee on the value of A_B is with respect to random $y = f(x), p$, whereas algorithm A_{f_1} doesn't do anything to "randomize" y: it's given as input to A_{f_1}. This difficulty is overcome by noting that, if A_B works well on random x, p, then, for a reasonably large fractions of x's, A_B continues to work well on these particular x's, and random p. We formalize this as follows.

Remember that $\mathrm{Adv}_{A_B} = \Pr\left[x, p \leftarrow \{0,1\}^k : A_B(f(x) \,.\, p) = x \odot p\right] \geq \epsilon + 0.5$. Let's call x good if $\mathrm{Adv}_{A_B} = \Pr\left[p \leftarrow \{0,1\}^k : A_B(f(x) \,.\, p) = x \odot p\right] \geq \epsilon/2 + 0.5$. A pretty fat fraction of x's have to be good:

Proposition 8.9.2 $\Pr\left[x \stackrel{R}{\leftarrow} \{0,1\}^k : x \text{ is good}\right] \geq \epsilon/2$.

This is because

$$
\begin{aligned}
\epsilon + 0.5 \;&\leq\; \Pr_{x,p}\left[A_B(f_1(x) \,.\, p) = x \odot p\right] \\
&\leq\; \Pr_p\left[A_B(f_1(x) \,.\, p) = x \odot p \mid x \text{ is good}\right] \cdot \Pr_x[x \text{ is good}] + \\
&\qquad \Pr_p\left[A_B(f_1(x) \,.\, p) = x \odot p \mid x \text{ is not good}\right] \cdot \Pr_x[x \text{ is not good}] \\
&\leq\; 1 \cdot \Pr_x[x \text{ is good}] + (0.5 + \epsilon/2) \cdot 1
\end{aligned}
$$

Thus $\epsilon + 0.5 \leq \Pr[x \text{ is good}] + 0.5 + e/2$, and so $\Pr[x \text{ is good}] \geq e/2$.

Pairwise-independence tail inequality. To proceed with the analysis, we will need the following probablity bound. As an exercise, prove it from the Chebyshev bound, which you can look up in any book on probablity.

Lemma 8.9.1 Let $\mathsf{X}_1, \cdots, \mathsf{X}_m$ be pairwise independent 0/1 random variables with common mean μ. Let $\mathsf{X} = \sum_i \mathsf{X}_i / m$. Let $\mu > 1/2$. Then

$$\Pr[|\mathsf{X} - \mu| \geq \lambda] \;\leq\; \frac{1}{4\lambda^2 m}.$$

Compare the above with the Chernoff bound we've used before. The rate of drop-off in the probablity is polynomial instead of exponential. But the independence assumption is much weaker: pairwise independence instead of complete independence.

A calculation. Now we know that an $e/2$ fraction of the x's are good, so let's assume, for the purpose of analyzing A_{f_1}, that x is good. Let's assume, next, that we've guessed the right values for $\beta_0, \cdots, \beta_t$. This happens with probablity 2^{-t}.

Fix $1 \leq j \leq k$. We observe that, since $\mathsf{P}_1, \cdots, \mathsf{P}_m$ are uniformly distributed and pairwise independent, the same holds for $\mathsf{P}_1^{(j)}, \cdots, \mathsf{P}_m^{(j)}$. Let X_i^j be indicator random variable which is 1 if $guess_i^j = x \odot p$, and 0 otherwise. Let $\mathsf{X}^j = \sum_i \mathsf{X}_i^j / m$. Because we've assumed that x is good, and we've also assumed that $\beta_{t-1} \cdots \beta_0$ are correct, we know that $\mathsf{X}_1^j, \mathsf{X}_2^j, \cdots, \mathsf{X}_m^j$ are pairwise independent with common mean $\mu = 0.5 + \epsilon/2$. Using the pairwise-independence tail inequality with $\lambda = \epsilon/2$ we have that

$$
\begin{aligned}
&\Pr\left[A_{f_1} \text{ errs in predicting } x[j] \mid x \text{ is good and } \beta_{t-1} \cdots \beta_0 \text{ are correct}\right] \\
&\quad \leq \Pr\left[|\mathsf{X} - \mu| \geq \epsilon/2\right] \\
&\quad \leq \frac{1}{\epsilon^2 m}
\end{aligned}
$$

Let's choose m large enough so that $\epsilon^2/2m$ will be less than $1/2k$. Then, continuing to assume that x is good and that we've guessed the correct $\beta_{t-1} \cdots \beta_0$, with probability at least $k \cdot (1/2k) = 0.5$ algorithm A_{f_1} will get *every* bit $x[j]$ correct and, in such a case, she will invert f. Thus we want

$$\frac{1}{\epsilon^2 m} \leq \frac{1}{2k}$$

so select

$$m = \left\lceil \frac{2k}{\epsilon^2} \right\rceil .$$

So: selecting m as above will result in A_{f_1} successfully inverting f_1 at y a fraction $0.5 \cdot 2^{-t} \cdot \epsilon/2 \approx \epsilon^3/(8k)$ of the time. We can get rid of the 2^{-t} by trying all the possible $\beta_{t-1} \cdots \beta_0$ values ourselves, since one of them must be correct. Doing this will add to the time of A_{f_1}, bringing it to $O(k^3 \epsilon^{-4} T_{A_B} + k\epsilon^{-2} T_{f_1})$, where T_{A_B} is the running time for a single call to A_B, and T_{f_1} is the time to compute f_1. The success probablity will be $\epsilon/4$.