CSA E0 235: Cryptography (29-September-2019)

Scribe for Lecture 13

Instructor: Arpita Patra *Submitted by: (Shravani Mahesh Patil)*

1 Do One Way Functions Exist?

The existence of one way functions (OWFs) is till date a conjecture and proving unconditionally that one way functions exist is hard. This is attributed to the fact that, the existence of one way functions would solve one of the most important open questions of theoretical computer science and prove that $P \neq NP$. The intuition behind this claim is that, as inferred from the definition of one way function, the one-wayness of a function implies that it is "hard" to invert the function in the average case. (Note here that the term "hard" indicates that there does not exist a known polynomial time algorithm to solve a problem.) Whereas, the "hardness" in complexity theory refers to the worst case. This implies that the one way function conjecture is stronger than $P \neq NP$. That is, if one way functions exist, then there exists a function which is "hard" to invert in the average case. This further implies that the function is "hard" to invert in the worst case. This function itself is then an instance of a problem which is in NP but not in P. Thus we can conclude that, Existence of one way functions $\implies P \neq NP$.

However, the converse of this is not true. That is, proving that $P \neq NP$, does not imply the existence of one way functions. As noted, $P \neq NP$ implies that there exists some problem for which at least one of the input instances does not have a polynomial time algorithm which can solve the problem. It is crucial to note that a single such input instance for any problem is enough to prove that $P \neq NP$. However, the existence of one way functions requires average case hardness. That is, for a uniformly random choice of inputs, the one way function is required to be hard to invert, which is not guaranteed by the proof of $P \neq NP$. Hence, $P \neq NP \implies$ Existence of one way functions.

We have already mentioned that the existence of one way functions is a stronger conjecture than that of $P \neq NP$. Consequently, we can also deduce that the non existence of one way functions, does not imply that $P = NP$. The intuitive argument behind is along the similar lines as described. The non existence of one way functions, implies that there does not exist a function which is hard to invert in the average case. However, $P \neq NP$ requires the hardness of a problem to hold only in the worst case. The non existence of one way functions does not ensure any behavior of a problem in the worst case and hence cannot imply that $P = NP$. Thus, we can conclude that, Non existence of one way functions $\implies P = NP$

However, the converse implication that $P = NP$ implies Non existence of one way functions holds. It is clear from $P = NP$ that there exists a polynomial time algorithm for every input instance of each problem. This implies that there cannot exist a function which is hard to invert in the average case, thus concluding that $P = NP \implies$ Non existence of one way functions.

Summarizing the above, we have:

- Existence of OWFs $\implies P \neq NP$
- $P \neq NP \not\implies$ Existence of OWFs
- Non existence of OWFs $\not\implies P = NP$
- $P = NP \implies$ Non existence of OWFs

2 Candidate One Way Functions

Although the existence of one way functions is a conjecture, it is based on the fact that there exist several computational problems which do not have any known polynomial time algorithm for solving them. In this section we enumerate two candidate one way functions and the corresponding hard problems they rely on.

- Candidate function based on the *Integer Factorization Problem*

  $$f_1(x, y) = x.y$$

  where x and y are large prime numbers of equal length. Given an output $z = f_1(x, y)$, inverting the function f_1 requires identifying x and y. The above candidate function relies on the integer factorization problem, that is, computing the product of two numbers is easy but given the number, it is difficult to compute its prime factors. As we have seen previously, if we do not impose any restriction on the domain of the function, then the function is not one way. For the function f to be a one way function, we thus restrict the domain to equal length prime numbers x and y.

- Candidate function based on the *Subset-Sum Problem*

  $$f_2(x_1, x_2, \ldots, x_n, J) = (x_1, x_2, \ldots, x_n, [\textstyle\sum_{j \in J} x_j \mod 2^n])$$

  where $\forall i \in \{1, 2, \ldots, n\}$, x_i is an n bit string, J is an n bit string which specifies a subset of $\{1, 2, \ldots, n\}$. Given an output $(x_1, x_2, \ldots, x_n, y) = f_2(x_1, x_2, \ldots, x_n, J)$, inverting the function f_2 requires the identifying a string $\hat{J}$ corresponding to a subset of $\{1, 2, \ldots, n\}$, such that $\sum_{j \in \hat{J}} x_j \mod 2^n = y$. Notice that, the underlying computational problem is $NP-$complete. However, $NP-$completeness indicates that there exists at least one instance of input for which there does not exist a polynomial time algorithm which can solve the *Subset-Sum Problem*. However, for the function f_2 to be a candidate one way function, it is required that there does not exist a polynomial time algorithm for the subset-sum problem in the average case. Thus the candidate one way function does not merely rely on the fact that the subset-sum problem is $NP-$complete, but rather it leverages on the fact that there does not exist a known polynomial time algorithm to solve the subset-sum problem on a random set of inputs.

2.1 One Way Permutations

Before proceeding further, we briefly define the concept closely related to one way functions, that is, one way permutations. To define one way functions, we first define length preserving functions.

1. *Length Preserving Function:* A function f is said to be length preserving, if for all x, $|f(x)| = |x|$. That is, the image and the preimage are of the same size.
2. *One Way Permutation:* A function f is a one way permutation if it satisfies the following [1].
   (a) f is a one way function.
   (b) f is has one-to-one mapping.
   (c) f is length a length preserving function.

Note that, in case of a one way permutation, although every $y = f(x)$ has a unique preimage x, it must be difficult to compute x given $f(x)$.

3 Does a One Way Function Not Reveal Any Information About Its Input?

From the definition of a one way function, we know that for a one way function f, given $f(x)$, it is hard to it is hard to invert f. This indicates that given $y = f(x)$, it is hard to compute x. This might give the impression that in a one way function, $f(x)$ does not leak any information about x. However, this is not necessarily true. The definition of a one way function guarantees only that the entire x cannot be computed from $f(x)$. Hence a partial leak of information regarding x is allowed by the definition of a one way function, as long as the entire x cannot be computed from $f(x)$ by an polynomial time algorithm. To ensure this, consider the following example of a one way function.

Let g be a one way function. We define a new function f as follows:

$$f(x_1, x_2) = (x_1, g(x_2))$$

where $|x_1| = |x_2|$. Note that f reveals half of its input. However, function f can be easily proven to be a one way function by reducing the one-wayness of g to the one-wayness of f using a simple reduction based proof. Hence it is clear that a one way function may leak some information about its input.

However, the one-wayness of a one way function intuitively implies that there exists some information regarding the input which is not leaked by the one way function from its output, which prevents the computation of the entire input in polynomial time. This information regarding the input which is hidden by the one way function is captured formally by the notion of *hard core predicates*.

4 Hard Core Predicates

As mentioned, informally, a hard core predicate of a function, captures the information about the input which is not leaked from its output. A hard core predicate hc of a function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is a boolean function $hc : \{0,1\}^* \rightarrow \{0,1\}$ such that given $f(x)$ it is difficult to compute $hc(x)$ with a significant advantage over randomly guessing it. Formally [2],

Definition 1 A function $hc : \{0,1\}^* \rightarrow \{0,1\}$ is a hard core predicate of a function f if hc can be computed in polynomial time, and for every probabilistic polynomial time algorithm $\mathcal{A}$ there is a negligible function $negl$ such that

$$\Pr_{x \leftarrow \{0,1\}^n} [\mathcal{A}(1^n, f(x)) = hc(x)] \leq \frac{1}{2} + negl(n)$$

where the probability is taken over the uniform choice of $x \in \{0,1\}^n$ and the randomness of $\mathcal{A}$. $\diamondsuit$

Note that the two functions f and its corresponding hc are defined over the same input space (domain). Also, the definition of a hard core predicate does not require the function f to be a one way function. Consider the following function

$$f(x) = f(x_1 x_2 \dots x_n) = x_1 x_2 \dots x_{n-1}$$

Define the predicate $hc(x) = x_n$, where $x = x_1 x_2 \dots x_n$. It is clear that $hc(x)$ is the hard core predicate for f, since. That is, given $f(x) = x_1 x_2 \dots x_{n-1}$, it is hard to determine x_n, since it is independent of the output. That is, $\Pr[x_n = 0 | f(x)] = \Pr[x_n = 1 | f(x)] = \frac{1}{2}$.

However, if f is a permutation and has a hard core predicate, then f must be one way. In other words, a permutation f cannot have a hard core predicate, unless it is one way. However, we focus our attention only on the hard core predicates corresponding to one way functions and one-way permutations.

4.1 Finding Hard Core Predicates for One Way Functions is Not Simple

We now establish the fact that, there does not exist a universal hard core predicate. In other words, given any predicate hc, there exists a one way function f, such that hc is not a hard core predicate for f. To concretely establish this fact, consider the following example of a predicate.

$$hc(x) = \oplus_{i=1}^{n} x_i$$

where $x = x_1 x_2 \dots x_n$ is an n bit binary string.

The intuition behind considering hc as a candidate hard core predicate of any one way function f, is that, since f is hard to invert, given $f(x)$, it must hide at least one of the bits of its preimage x. This would imply that computing $\oplus_{i=1}^{n} x_i$ is hard. However, this argument regarding hc being a universal hard core predicate is incorrect. To prove this, consider a one way function g and define a new one way function f as follows:

$$f(x) = (\oplus_{i=1}^{n} x_i, g(x))$$

where $x = x_1 x_2 \dots x_n$ is an n bit binary string. By the same argument as that outlined in Section 3, we can prove that the function f is a one way function. However, note that $hc(x) = \oplus_{i=1}^{n} x_i$ is not a hard core predicate for f, since the output of f itself reveals $\oplus_{i=1}^{n} x_i$. In fact, a similar argument can be applied for any given predicate hc and a corresponding one way function can be designed for which hc is not a hard core predicate. This indicates that there does not exist a universal hard core predicate.

4.2 Do Hard Core Predicates Exist for Any One Way Function?

It remains an open question whether a hard core predicate exists for any one way function or not. However, our requirement for building the cryptographic primitives is satisfied by a weaker assumption. Specifically, we show that given a one way function f, we can construct another one way function g and its corresponding hard core predicate hc.

Theorem 1 ***(Goldreich-Levin Theorem)****: Assume one way functions (resp., permutations) exist. Then there exists a one way function (resp., permutation) g and a hard core predicate hc of g [2].*

Let f be a one way function. Function g and its corresponding hard core predicate hc are constructed as follows:

$$g(x, r) = (f(x), r)$$

where $|x| = |r|$ and

$$hc(x, r) = \oplus_{i=1}^{n} x_i \cdot r_i$$

where $x = x_1 x_2 \dots x_n$ and $r = r_1 r_2 \dots r_n$ are n bit binary strings.

To prove the above theorem, we have to prove the following two statements:

1. g is a one way function. (The proof follows a reduction argument as outlined in Section 3 and is left as a reading assignment.)

2. If f is a one way function then hc is a hard core predicate for the one way function g.

To prove the second statement, we prove its contrapositive. That is, if there exists an adversary $\mathcal{A}$ which can compute $hc(x, r)$ given $g(x, r)$ with non-negligible advantage, then we can construct an adversary $\mathcal{A}'$ which can invert f. That is, if there exists an adversary $\mathcal{A}$ such that

$$\Pr_{x,r \leftarrow \{0,1\}^n} [\mathcal{A}(f(x), r) = hc(x, r)] \geq \frac{1}{2} + \frac{1}{p(n)}$$

then we can construct an adversary $\mathcal{A}'$ such that,

$$\Pr_{x \leftarrow \{0,1\}^n} \left[\mathcal{A}'(1^n, f(x)) = x\right] \geq \frac{1}{p'(n)}$$

Due to the complexity involved in proving the above statement, we first prove two weaker claims.

Claim 2 *If there exists an adversary $\mathcal{A}$ such that,*

$$\Pr_{x,r\leftarrow\{0,1\}^n}[\mathcal{A}(f(x),r) = hc(x,r)] \geq \frac{3}{4} + \frac{1}{p(n)}$$

then we can construct an adversary $\mathcal{A}'$ such that,

$$\Pr_{x\leftarrow\{0,1\}^n}\left[\mathcal{A}'(1^n, f(x)) = x\right] \geq \frac{1}{4p(n)}$$

Claim 3 *If there exists an adversary $\mathcal{A}$ such that,*

$$\Pr_{\forall x,r\leftarrow\{0,1\}^n}[\mathcal{A}(f(x),r) = hc(x,r)] = 1$$

then we can construct an adversary $\mathcal{A}'$ such that,

$$\Pr_{\forall x\leftarrow\{0,1\}^n}\left[\mathcal{A}'(1^n, f(x)) = x\right] = 1$$

Proof We start by proving Claim 3. We assume that there exists a PPT adversary $\mathcal{A}$ which, given $g(x,r) = (f(x),r)$, can compute $hc(x,r)$ for all $x, r \in \{0,1\}^n$. We construct an adversary $\mathcal{A}'$ which, given $f(x)$ can invert f by leveraging on the adversary $\mathcal{A}$ as follows.

1. For each $i \in \{1, 2, \ldots, n\}$,
   (a) Construct the string e^i as an n bit binary string with 1 in the ith bit position and 0 in all other bit positions.
   (b) Pass to $\mathcal{A}$, the challenge $(f(x), e^i)$.
   (c) Let $\hat{x}_i = \mathcal{A}(f(x), e^i)$.
2. Output $\hat{x} = \hat{x}_1\hat{x}_2\ldots\hat{x}_n$

We now show, that $\hat{x} = x$. That is, we show that, $\hat{x}_i = x_i$, $\forall i \in \{1, 2, \ldots, n\}$. Observe that,

$$\hat{x}_i = \mathcal{A}(f(x), e^i) = hc(x, e^i) = \oplus_{j=1}^{n}\, x_j \cdot e^i_j = x_i$$

The above equality holds since, e^i is a string with 1 in only its ith bit position. This indicates that, only x_i will be considered in the exclusive-or, whereas the terms $x_j \cdot e^i_j$, $\forall j \neq i$ will be 0. It is thus clear that, with n queries to the adversary $\mathcal{A}$, the newly constructed adversary $\mathcal{A}'$ can retrieve x corresponding to its challenge $f(x)$. That is,

$$\Pr_{\forall x\leftarrow\{0,1\}^n}\left[\mathcal{A}'(1^n, f(x)) = x\right] = 1$$

However, this is a contradiction. Since, we know that f is a one way function, the existence of $\mathcal{A}'$ is a contradiction which arose due to the underlying assumption of the existence of an adversary $\mathcal{A}$ which can always compute $hc(x,r)$ given $g(x,r)$. Hence, there does not exist a PPT adversary, which can always compute $hc(x,r)$, given $g(x,r) = (f(x),r)$. ■

The proof of Claim 2 requires a stronger argument and is more complex than that of Claim 3. The same proof strategy that we have applied to prove Claim 3, is not applicable for proving Clam 2. This is due to the following reasons:

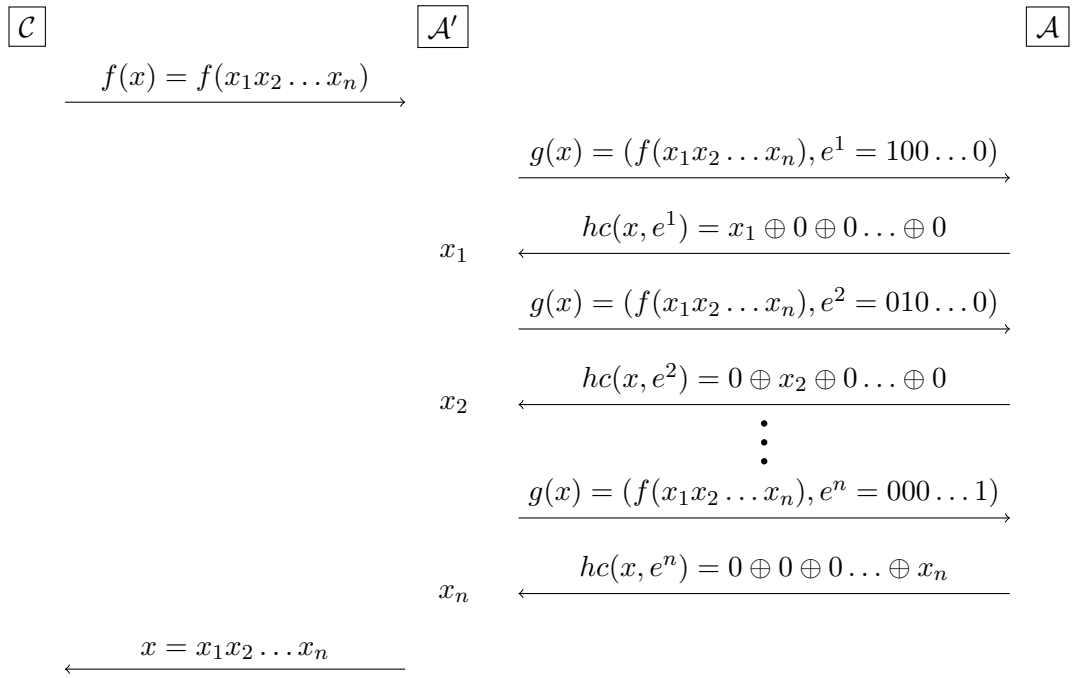

Figure 1: Adversarial Game for Proof of Claim 3

1. The adversary $\mathcal{A}$ for Claim 3 is guaranteed to output the correct value of $hc(x, r)$ for all x and r. However, this is not guaranteed by the adversary $\mathcal{A}$ considered in the statement of Claim 2, who has a probability $\geq \frac{3}{4} + \frac{1}{p(n)}$ of computing $hc(x, r)$ correctly for x, r chosen uniformly at random. Thus, it is possible that the adversary always outputs the incorrect $hc(x, r)$, suppose when $r = 100 \ldots 0$. This would further imply that the adversary $\mathcal{A}'$ cannot compute x_1 correctly.

2. Secondly, it is crucial to note that for the adversary $\mathcal{A}$ from the statement of Claim 2, the interaction with $\mathcal{A}'$ will emulate the adversarial game correctly, if and only if x and r are chosen uniformly at random. Whereas, in proving Claim 3, the adversary $\mathcal{A}'$ was not required to follow any distribution in choosing the values of r was allowed to choose the values of r to be $e^1 = 100 \ldots 0$ and so on upto $e^n = 000 \ldots 1$. Clearly, this choice of r values is not done uniformly at random. Thus, following the same strategy for proving Claim 2, will not emulate the game correctly for the adversary $\mathcal{A}$.

These two reasons make it clear that to prove Claim 2, a different argument is necessary. The proof of this claim will follow in the subsequent lecture. However, in the next section, we discuss the construction of a pseudorandom generator by assuming that the claims are correct and that the required proofs have been established.

5 Pseudorandom Generator With Minimal Expansion

Given a one way permutation f with a corresponding hard core predicate hc, we can construct a pseudorandom generator defined as $G(s) = f(s)||hc(s)$. The intuition behind $G(s)$ being a pseudorandom generator is as follows. Since, f, is a one way permutation, its output $f(s)$ is distributed uniformly when s is chosen uniformly at random. Moreover, since $hc(s)$ is the hard core predicate of f, it is difficult to distinguish $hc(s)$ from a random bit with non-negligible advantage. This intuitively implies that the string $f(s)||hc(s)$ is pseudorandom when s is chosen uniformly at random from the input space. We now prove this claim formally.

Theorem 4 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation with hard core predicate $hc : \{0,1\}^n \to \{0,1\}$. Then algorithm G, defined by $G(s) = f(s)||hc(s)$ is a pseudorandom generator with expansion factor $l(n) = n + 1$.*

Proof To prove that $G(s)$ is a pseudorandom generator, we will prove the contrapositive of the above statement. Specifically, we prove that, if there exists a PPT distinguisher D for G having non-negligible advantage, then we can construct a PPT adversary $\mathcal{A}$ which, given $f(s)$, can compute the hard core predicate $hc(s)$ of the one-way permutation f with non-negligible advantage.

Let D be the PPT distinguisher for G. That is,

$$\Pr_{r\leftarrow\{0,1\}^{n+1}}[D(r) = 1] - \Pr_{s\leftarrow\{0,1\}^n}[D(G(s)) = 1] \geq \frac{1}{p(n)} \tag{1}$$

We first show that, such a distinguisher D, can also distinguish between strings of type $f(s)||hc(s)$ and $f(s)||\overline{hc}(s)$. To see this, it is crucial to note that, when s is uniformly chosen, the string $f(s)$ is uniformly distributed and hence is a random n bit string. Keeping this in mind, we have,

$$\begin{aligned}
\Pr_{r\leftarrow\{0,1\}^{n+1}}[D(r) = 1] &= \Pr_{r\leftarrow\{0,1\}^n, r'\leftarrow\{0,1\}}\left[D(r\|r') = 1\right] \\
&= \Pr_{s\leftarrow\{0,1\}^n, r'\leftarrow\{0,1\}}\left[D(f(s)\|r') = 1\right] \\
&= \frac{1}{2}\cdot\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|hc(s)) = 1] + \frac{1}{2}\cdot\Pr_{s\leftarrow\{0,1\}^n}\left[D(f(s)\|\overline{hc}(s)) = 1\right]
\end{aligned} \tag{2}$$

It is also clear from the definition of G that,

$$\Pr_{s\leftarrow\{0,1\}^n}[D(G(s)) = 1] = \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|hc(s)) = 1] \tag{3}$$

From Equations (1),(2) and (3), we can observe that,

$$\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}\left[D(f(s)\|\overline{hc}(s)) = 1\right] - \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|hc(s)) = 1]\right) \geq \frac{1}{p(n)} \tag{4}$$

We now show that, we can construct the adversary $\mathcal{A}$ using the distinguisher D as a subroutine. Note that, $\mathcal{A}$ receives as its challenge the value $y = f(s)$ and requires to successfully compute $hc(s)$. The adversary $\mathcal{A}$ follows the subsequent steps to leverage on the distinguisher D to successfully compute $hc(s)$.

1. Chooses a random $r' \in \{0,1\}$.
2. Passes on to D, the challenge $y||r'$.
3. If D outputs 0, output $hc(s) = r'$, otherwise output $hc(s) = \overline{r}'$.

(**Note**: The convention used here is that D outputs a 0, for the strings it identifies to be of type $f(s)||hc(s)$ and outputs a 1 for the string it identifies to be of type $f(s)||\overline{hc}(s)$.)

We can see that,

$$\begin{aligned}
\Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s)) = hc(s)] &= \frac{1}{2}\cdot \Pr_{s\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(s)) = hc(s)|r' = hc(s)\right] \\
&\quad + \frac{1}{2}\cdot \Pr_{s\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(s)) = hc(s)|r' \neq hc(s)\right] \\
&= \frac{1}{2}\cdot \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|hc(s)) = 0] + \frac{1}{2}\cdot \Pr_{s\leftarrow\{0,1\}^n}\left[D(f(s)\big\|\overline{hc}(s)) = 1\right] \\
&= \frac{1}{2}\cdot\left(\left(1 - \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|hc(s)) = 1]\right) + \Pr_{s\leftarrow\{0,1\}^n}\left[D(f(s)\big\|\overline{hc}(s)) = 1\right]\right) \\
&= \frac{1}{2} + \frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^n}\left[D(f(s)\big\|\overline{hc}(s)) = 1\right] - \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|hc(s)) = 1]\right) \\
&\geq \frac{1}{2} + \frac{1}{p(n)} \qquad \text{(From Equation (4))}
\end{aligned}$$

However, this is a contradiction. Since hc is a hard core predicate for f, we know that

$$\Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s)) = hc(s)] \leq \frac{1}{2} + negl(n)$$

Hence, our assumption of a distinguisher D for G is incorrect, thus proving that G is a pseudorandom generator.

■

References

[1] A. Patra. Csa e0_235 : Cryptography (august - december 2019). [Online]. Available: https://www.csa.iisc.ac.in/ cris/e0_235.html

[2] J. Katz and Y. Lindell, *Introduction to modern cryptography.* Chapman and Hall/CRC, 2014.