Notes for Lectures 12–14

1 General One-Way and Trapdoor Functions

In this section, we will try to generalize what we've seen so far. For example, we know how to build a secure encryption out of RSA, but what exactly is RSA itself? In modern terms, it is a trapdoor permutation family, which we define below.

1.1 One-Way Functions

Let us first introduce one-way functions. We've actually seen concrete examples of them before; this is just a generalization, so we can talk of a one-way function f independent of its particular implementation.

Definition 1. A function $f : \{0,1\}^* \to \{0,1\}^*$ is one-way if

1. it is polynomial-time computable;
2. it is hard to invert, i.e., for all probabilistic polynomial-time A there exists a negligible function η such that, for all k, $\Pr[f(A(f(x),1^k)) = f(x)] \le \eta(k)$, where the probability is taken over a random choice of k-bit string x and coin tosses of A.

Note that it's important that we are not requiring A to find x; rather, any inverse of $f(x)$ is fine. Of course, if f is a permutation (i.e., a bijective function), then it would be equivalent to require A to find x, because x is the only inverse of $f(x)$.

Note also the importance of selecting the input to A: the input is not selected uniformly at random; rather, x is selected uniformly at random, and the input is $f(x)$. Of course, again, if f is a permutation, then the two are equivalent.

An example is the following f: split the k-bit input into strings a of length $\lfloor k/2 \rfloor$ and b of length $\lceil k/2 \rceil$, and output $c = ab$. The inverter A would have to find two *large* factors of c, which is believed to be hard. Note that the input c of A is not a uniformly selected integer; in particular, we know that it has two factors of (nearly) the same length.

The existence of one-way functions is the minimal assumption necessary (though often not sufficient) for almost anything interesting in cryptography. Note that the assumption that one-way functions exist is stronger than the assumption that P≠NP (intuitively, because one-way functions are hard on the average case, where as it could be that NP-complete problems are hard only very infrequently).

A *one-way permutation* is a one-way function that is a bijection of $\{0,1\}^k$ to $\{0,1\}^k$ for each k.

1.2 One-Way Function Families

The examples we've seen in class, such as modular squaring, RSA, and Discrete Logarithm, are not quite one-way functions by the above definition. Rather, they are one-way function families, as defined below.

Definition 2. Let I be an index set. A collection of functions $\{f_i : D_i \to R_i\}_{i \in I}$ is called one-way, if:

1. there exists a probabilistic polynomial-time algorithm Gen that, on input 1^k, picks $i \in I$;
2. there exists a probabilistic polynomial-time algorithm M that, on input $i \in I$, picks $x \in D_i$;
3. given i and x, the value $f_i(x)$ is polynomial-time computable;

4. for all probabilistic polynomial-time A there exists a negligible function η such that, for all k, if i is chosen by $\mathrm{Gen}(1^k)$ and x is chosen by $M(i)$, $\Pr[f_i(A(f_i(x), i, 1^k)) = f_i(x)] \leq \eta(k)$, where the probability is taken over coin tosses of Gen, M and A.

For example, for Discrete Logarithm, the index set $I = \{(p, g) | p$ is prime $, g$ is a generator of $\mathbb{Z}_p^*\}$, and for $(p, g) \in I$, $D_{(p,g)} = R_{(p,g)} = \mathbb{Z}_p^*$ and $f_{(p,g)}(x) = g^x \bmod p$.

A *collection of one-way permutations* is a collection of one-way functions with the additional property that f_i is a permutation. The discrete logarithm collection is actually a collection of one-way permutations.

1.3 Trapdoor Permutations

A collection of one-way permutations with the additional property that the (unique) inverse is easy to obtain with some special information is called a collection of trapdoor permutations.

Definition 3. A collection of one-way permutations $\{f_i : D_i \to R_i\}_{i \in I}$ is called *trapdoor* if there exists a probabilistic polynomial-time algorithm Inv and if Gen, in addition to outputting $i_{\in} I$, outputs a value t with the following property: for all $x \in D_i$, $\mathrm{Inv}(t, f_i(x)) = x$.

For example, RSA is a collection of trapdoor permutations. The index set consists of pairs (n, e); the trapdoor information t is (n, d); and the domain and the range are $\mathbb{Z}_n^*$.

1.4 Generalizing Results

To obtain a pseudorandom generator, both the Blum-Micali and the Blum-Blum-Shub generators simply selected a one-way permutation from a family, and iterated it multiple times on a random initial seed, each time outputting a bit that's hard to predict. It is natural to ask whether for any one-way permutation (family) there is such a bit. The following theorem of Goldreich and Levin answers this question in the affirmative. We state it somewhat informally, and do not prove it here.

Theorem 1 ([GL89]). *Let f be a one-way function (the same also holds for families of one-way functions). Let r be a random k-bit value. Then, for a random k-bit x, the bit $r \cdot x$ is hard to compute with probability greater than $1/2$, given $f(x)$ and r. (Here $r \cdot x = r_1 x_1 \oplus r_2 x_2 \oplus \ldots r_k x_k$, the inner-product modulo 2 of r and x.)*

Therefore, our constructions of pseudorandom generators extend to *any* one-way permutation f (and, similarly, one-way permutation family). We simply take our seed to be (x, r), let $x_0 = x, x_i = f(x_{i-1})$, and output the bits $b_i = x_i \cdot r$.

Hence, we get

Theorem 2. *If one-way permutations (or families) exist, then so do pseudorandom generators.*

However, one-way functions are a weaker assumption, and it would be nice to know if pseudorandom generators can be based on just one-way functions, not permutations. The following theorem of Håstad, Impagliazzo, Levin and Luby shows that one-way functions suffice. It is quite difficult to prove.

Theorem 3 ([HILL99]). *Pseudorandom generators exist if and only of one-way functions exist.*

Thus, one-way functions suffice for symmetric encryption. However, they do not suffice for public-key encryption: you really need the trapdoor to be able to go back. Note also that by generalizing our previous two bit-by-bit constructions, we know that trapdoor permutations suffice.

Finally, I want to mention two constructions of Levin's [Lev87, Lev03] that address the existence of one-way functions. In both, he constructs a single function U with the following property: U is one-way if one-way functions exist. U is known as the *universal one-way function*. The question of whether one-way functions exist reduces to the question of whether this specific single function is one-way.

2 Diffie-Hellman Key Exchange

A great surge of academic interest in modern cryptography started with the work of Diffie, Hellman, and Merkle, and the publication of "New Directions in Cryptography" by Diffie and Hellman [DH76]. In this work, Diffie and Hellman proposed the idea of public-key encryption and digital signatures. Although they didn't have an implementation of public-key encryption, they did suggest something close, called "key agreement."

Here is the idea. Suppose there is a fixed prime p and generator g of $\mathbb{Z}_p^*$ known to everyone. Alice and Bob want to agree on a secret they can both use for some symmetric encryption scheme. To do so, Alice selects a random $a \in \mathbb{Z}_p^*$ and sends $g^a \bmod p$ to Bob. Bob similarly selects a random $b \in \mathbb{Z}_p^*$ and sends $g^b \bmod p$ to Alice. Now Alice can compute $K = g^{ab}$ by raising g^b to the power a, and Bob similarly can compute K by raising g^a to the power b. It is believed that g^{ab} is hard to compute from just g, g^a and g^b. More formally, this is known as the Computational Diffie-Hellman Assumption.

Assumption 1. For any poly-time algorithm A, there exists a negligible function η such that, if you generate random k-bit prime p and its generator g, and select a random $a, b \in \mathbb{Z}_p^*$, $\Pr[A(p, g, g^a \bmod p, g^b \bmod p) = (g^{ab} \bmod p)] \le \eta(k)$.

Note that if p and g are not known to both parties in advance, Alice can simply send both to Bob together with g^a.

3 A Bit More History

In 1977, the RSA cryptosystem [RSA78] appeared in Scientific American, helping generate public interest in the subject.

Until 1976, research in cryptography was mostly done in classified research labs, such as the National Security Agency in the United States, for military and intelligence purposes. Documents declassified by the UK in the late 1990s and now available on the web [Ell87] showed that public-key cryptography in general, and Diffie-Hellman and RSA specifically, were discovered in the classified community before their discovery in academia. Specifically, in 1970, James H. Ellis [Ell70] proposed the idea of public-key cryptography, which he termed "non-secret encryption"; in 1973, Clifford C. Cocks [Coc73] proposed RSA (although Cocks suggested using specific public exponent n, equal to the modulus, rather than a more general public exponent); and in 1974, Malcolm J. Williamson [Wil74, Wil76] proposed what we know as Diffie-Hellman. It's worth noting that the discoveries of RSA and Diffie-Hellman occurred in reverse order in the classified community, and that neither preceded the academic discoveries by more than a few years. It seems (according to what we know) that there wasn't much interest in public-key encryption in the military and intelligence community. One possible reason is that with rigid command structures such as those in the military, it is easy enough to establish shared secret keys (public-key ideas are of great help when people who have never seen each other before want to talk; this doesn't happen too much in the military). The second commonly cited reason is that the state of computers in the 1970s did not allow for such expensive operations as modular exponentiation to be easily carried out "in the field."

4 Man-in-the-middle attack against Diffie-Hellman

Imagine now that an adversary Eli is capable of not only intercepting messages between Alice and Bob, but also stopping them and substituting his own messages instead. Then Eli can do the following: pick his own random $e \in \mathbb{Z}_p^*$, and compute $g^e \bmod p$. Then intercept g^a that Alice sends to Bob, and substitute g^e instead. Note that Bob doesn't notice any difference (because, after all, both g^a and g^e are random elements of $\mathbb{Z}_p^*$), and dutifully replies with g^b. Eli intercepts g^b, and sends g^e to Alice instead. This way, Alice ends up thinking that she is sharing $K_1 = g^{ea}$ with Bob, while Bob ends up thinking that he is sharing $K_2 = g^{eb}$ with Alice. Note that, in fact, they are both sharing a key with Eli, who can compute g^{ea} and g^{eb}. Now whenever Bob tries to send something to Alice, he'll presumably encrypt (and/or authenticate) it using K_2. Eli can intercept it, decrypt with K_2, reencrypt with K_1, and send it on to Alice. So Bob and Alice will never realize they aren't sharing a key with each other.

This is known as "man-in-the-middle" attack, and is just one of the reasons why key agreement is a difficult problem. In fact, satisfactory formal definitions for key agreement took about a decade and a half longer to appear than definitions for encryption and signature. We will not study key agreement in this class. We will, however, use Diffie-Hellman below.

5 ElGamal Encryption

Taher ElGamal [ElG85] proposed the following way to make Diffie-Hellman into an encryption scheme. Alice publishes $p, g, g^a \bmod p$, as a public key, and keeps a as the secret key. To encrypt a message $m \in \mathbb{Z}_p^*$, Bob picks $b \in \mathbb{Z}_p^*$ at random, computes $g^b \bmod p$, $K = g^{ab} \bmod p$, and $c = mK \bmod p$, and outputs $(c, g^b \bmod p)$. To decrypt, Alice computes K using g^b and a, and recovers m from mK by dividing.

The scheme as described above is not semantically secure, because there exists a distinguisher D with good probability of success. Here is how D works: it outputs two messages m_0 and m_1, such that $m_0 \in QR_p$ and $m_1 \notin QR_p$. Then, upon receiving the ciphertext $(c, g^b \bmod p)$, D checks if $c \in QR_p$ (by checking whether $c^{(p-1)/2} \bmod p$ is 1 or -1). If so, it outputs 1; else it outputs 0. Note that $K \in QR_p$ if and only if ab is even, i.e., with probability 3/4. Therefore, if $m \in QR_p$, then $mK \in QR_p$ with probability 3/4; if $m \notin QR_p$, then $mK \in QR_p$ with probability 1/4 (because a non-square times a non-square is a square, but a non-square times a square is a non-square). Hence, the difference of the probabilities of D's output being 1 on encryption of m_0 and encryption of m_1 is $3/4 - 1/4 = 1/2$, which is not negligible.

However, ElGamal scheme can be fixed if we restrict our attention not the entire group $\mathbb{Z}_p^*$, but rather to the subgroup of squares QR_p. If this subgroup is of prime order (i.e., if $(p-1)/2$ is a prime), then p is often called a *safe* prime (and $(p-1)/2$ a *Sophie Germain* prime). Then the following assumption is believed to hold.

Assumption 2. For any poly-time algorithm A, there exists a negligible function η such that, if you generate random k-bit safe prime $p = 2q+1$ for prime q, and select a random generator g of QR_p, and random integers a, b and c between 1 and q,

$$\begin{aligned} &|\Pr[A(p, g, g^a \bmod p, g^b \bmod p, g^{ab} \bmod p) = 1] - \\ &\quad \Pr[A(p, g, g^a \bmod p, g^b \bmod p, g^c \bmod p) = 1]| \quad \leq \quad \eta(k). \end{aligned}$$

This is known as the Decision Diffie-Hellman (DDH) assumption, because it states that it's hard to decide whether you got g^{ab} or g^c for a random c. Note that this is a much stronger assumption than Computational Diffie-Hellman (CDH): CDH states that it's hard to compute g^{ab}, while DDH states that not only is it hard to compute, it actually looks random. There are many who are uncomfortable with such a strong assumption.

Let us now reformulate ElGamal encryption to take advantage of DDH. Alice publishes as her public key $p = 2q + 1$, where q is prime; g of order q, which is a generator of QR_p; and $g^a \bmod p$, for a random a between 1 and q. She keeps a as her secret key. To encrypt a message m, $1 \leq m \leq q$, Bob picks b, $1 \leq b \leq q$ at random, computes $g^b \bmod p$, $K = g^{ab} \bmod p$, and $c = m^2 K \bmod p$ and outputs $(c, g^b \bmod p)$. To decrypt, Alice computes K using g^b and a, and recovers m^2 from $m^2 K$ by dividing. She then finds m by taking a square root (note that there are two square roots, but one is greater than $q = (p-1)/2$, so she knows which one is m).

Theorem 4. *The above cryptosystem is polynomially secure under the DDH assumption.*

The proof, which is not presented in full detail here, is by hybrid argument: one proves that encryption of any message m is indistinguishable from a random pair (g^c, g^b). This follows easily from the DDH assumption. Therefore, encryptions of m_0 and m_1 are indistinguishable.

6 Semantic Security

Recall that for information-theoretic encryption, we had two definitions of security. Shannon secrecy focused on just two messages (much like indistinguishability we defined for public-key encryption), and perfect secrecy focused on obtaining information from encryption of a single messages drawn at random from some distribution. This section defines the analogue of perfect secrecy for public-key encryption.

First of all, because we are interested in computational security, which is usually formulated in terms of asymptotics, we will have multiple distributions on the message space—one for each value of the security parameter k (Shannon didn't have to do this and could consider a single fixed message space, because he had no computational hardness requirements; we can do the same if we formulate everything in terms of concrete security for a particular k, as explained in the lecture on defining next-bit unpredictability for pseudorandom generators). We will restrict messages to be of length polynomial in k, and, because encryption cannot hide length, we will provide the adversary with information on the length of the message chosen. Secondly, we can't require that there should be no information about the plaintext in the ciphertext (of course there will be—in fact, the ciphertext, combined with the public key, uniquely determines the plaintext). Rather, we will say that this information is not usable in polynomial time: whatever function of the plaintext you can compute with the ciphertext you can also compute without it. An finally, we will give the adversary arbitrary auxiliary information it wants about the plaintext (this models information adversary could obtain by other means, such as observing the behavior of various parties, etc.).

More precisely, let S be a randomized function that generates messages given the security parameter k; we require that there exists some polynomial p such that $|S(k)| < p(k)$. Note that we do not require S to be efficiently computable, or even computable at all. This is meant to model the distribution of message that the encryptor wants to send. Let $f, h : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$ be functions (not necessarily computable) that take the security parameter k and the message as inputs, and output some string whose length is polynomial in k (i.e., there must exist q such that $|f(k,m)| \le q(k)$ and $|h(k,m)| \le q(k)$). These are meant to model the information that the adversary is interested in, and the information that the adversary already has, respectively. Finally, let A be a probabilistic polynomial-time machine that attempts to compute $f(k,m)$ given $h(k,m)$, the length of m, a public key, and an encryption of m using the public key. We want to say that there is a machine B that computes $f(m)$ without the encryption (thus, only from $h(k,m)$ and the length of m). Consider the following two experiments.

`expA`(k)
1. $m \leftarrow S(k)$
2. $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathrm{Gen}(1^k)$
3. $c \leftarrow \mathrm{Enc}_{\mathrm{PK}}(m)$
4. $x \leftarrow A(1^k, h(k,m), 1^{|m|}, c, \mathrm{PK})$
3. Output 1 if $f(k,m) = x$ and 0 otherwise

`expB`(k)
1. $m \leftarrow S(k)$
2. $x \leftarrow B(1^k, h(k,m), 1^{|m|})$
3. Output 1 if $f(k,m) = x$ and 0 otherwise

Note that B gets no information at all beyond what h and the length of the message reveal (and h, can, in particular, be the constant function that reveals no information). This is exactly the point of secure encryption: without any information you can compute f just as well as with the ciphertext and the public key.

Definition 4. A public-key encryption scheme $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ is *semantically secure* if for all probabilistic polynomial-time algorithms A there exists a probabilistic polynomial-time algorithm B with the following property: for all S, f, h , there is negligible function η such that for all k,

$$\Pr[\mathtt{expA}(k) \rightarrow 1] - \Pr[\mathtt{expB}(k) \rightarrow 1] \leq \eta(k).$$

This definition is originally due to [GM84]. There are many variations of it; this particular version follows [Gol04].

Theorem 5 ([GM84]). *A cryptosystem is semantically secure if and only if it is polynomially secure.*

The proof is not nearly as simple as in the information-theoretic case (we will not do it here; see [GM84] for the original proof and [DR98] for a simpler one; [Gol04] has all the details); in fact, the result is surprising to many. There are other definitions of security that turn out to be equivalent to this one, which shows that our understanding of the security of encryption is quite robust.

Semantic security helps prove various cryptographic constructs that use encryption as part of a larger protocol. It can be much more powerful than indistinguishability when used in proofs, because it essentially says that no interesting function of the plaintext can be computed by the adversary.

7 Public-key encryption in the real world

Most of encryption that actually happens in daily life is symmetric, not public-key. For example, banks and ATMs rely mostly on the symmetric cipher DES (which we will discuss eventually, but only briefly). Even when public-key encryption is used, it is used only to encrypt a symmetric key, which is then used to encrypt bulk data, because symmetric techniques are much faster than public-key ones.

As far as algorithms used in practice, the most popular one is by far RSA, and the second is ElGamal. Neither is used exactly as we studied it.

In fact, the most common way to use RSA until recently has been a standard known as PKCS #1 version 1.5 [RSA93]. To encrypt a message m, it specifies that one should pad it to the length of the modulus by prepending a zero byte, byte of value 2, at least eight (and as many as needed) random non-zero bytes, followed by another zero byte to separate the pad from the message itself. The resulting bit string gets exponentiated to the public exponent e modulo n.

There is little one can prove about this scheme, although recently Jonsson and Kaliski [JK02] proved its security in certain applications under a relatively strong assumption. At some point it was believed to be not only polynomially secure, but, in fact, secure even against chosen-ciphertext attacks. However, Bleichenbacher [Ble98] found a reasonably practical chosen-ciphertext attack against it. At that time, version 2.0 of PKCS #1 was in the works; currently the most recent version is 2.1. Both 2.0 and 2.1 can be proven not only semantically secure, but also secure against chosen-ciphertext attacks, in a special (unrealistic) model known as "random oracle model." Whether a proof in such a model is actually meaningful is a matter of some debate; we'll consider this subject later in the course. It seems that PKCS encryption is the most common standard used today.

Most problems in implementing encryption, however, do not come from considerations of provability. Rather, they come from we often dismiss as "implementation issues." I identified three of them in class

1. Randomness. Computers, cell phones, ATMs, etc., generally do not come equipped with good sources of random bits that would be unpredictable to the adversary. As we know, though, secret randomness is necessary for key generation and encryption.

2. Secrets. They are hard to keep secret. Today's popular operating systems tend not to provide ways of storing a secret in such a way that it is accessible only to authorized programs and to no one else. A common approach is store a secret encrypted with a password known only to the user. Unfortunately, users are terrible and remembering high-entropy passwords; in addition, the secret is vulnerable when it's decrypted and actually used in a computation.

3. Keys. As emphasized above, it's very important to authentically know the public key of the person you are sending the message to. There are some approaches to this problem we will discuss later in the course, but they all have drawbacks.

7.1 A warning about terminology

In the academic world, "public" key and "secret" key usually form a pair. In the commercial world, the name of the second component is often "private" key (which doesn't abbreviate nicely, where as $(\mathrm{PK}, \mathrm{SK})$ does). This wouldn't be too much of a problem, except that the commercial world also often uses "secret key" to mean "non-public key," such as DES, one-time-pad, etc. To avoid confusion, we will call things like DES and the one-time-pad "symmetric" cryptography (because both parties share the same key). (To further compound the confusion, some people use the term "private-key cryptography" to mean "symmetric cryptography".)

8 Man-in-the-middle attack against encryption

Note that man-in-the-middle attack also applies to encryption. If Bob wants to send something to Alice, and the two never met before, then Alice needs to send Bob her PK_A. If Eli intercepts it and substitutes his own PK_E instead, Bob won't know the difference. He will now encrypt his message to Alice using PK_E, thus allowing Eli to read it.

In other words, while public-key encryption removes the need to share keys secretly, it does not remove the need for sharing them *authentically*. Bob need not keep PK_A secret, but he does need to know that it came from Alice. We'll address this problem in the next lecture.

References

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 23–27 August 1998.

[Coc73] Clifford C. Cocks. A note on non-secret encryption, 1973. Available from `http://www.cesg.gov.uk/publications/index.htm`.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[DR98] Yevgeniy Dodis and Matthias Ruhl. A simple proof that GM-security → semantic security, 1998. Available from theory.lcs.mit.edu/˜yevgen/ps/proof.ps.gz.

[ElG85] Taher ElGamal. A public-key cryptosystem and a signature scheme based on the discrete logarithm. *IEEE Transactions of Information Theory*, 31(4):469–472, 1985.

[Ell70] James H. Ellis. The possibility of non-secret encryption, 1970. Available from `http://www.cesg.gov.uk/publications/index.htm`.

[Ell87] James H. Ellis. The story of non-secret encryption, 1987. Available from `http://www.cesg.gov.uk/publications/index.htm`.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15–17 May 1989.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[HILL99] J. Håstad, R. Impagliazzo, L.A. Levin, and M. Luby. Construction of pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[JK02] Jakob Jonsson and Burton S. Kaliski, Jr. On the security of RSA encryption in TLS. In Moti Yung, editor, *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 127–142. Springer-Verlag, 18–22 August 2002.

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[Lev03] Leonid A. Levin. The tale of one-way functions. *Problems of Information Transmission (Problemy Peredachi Informatsii)*, 39(1):92–103, 2003. Available at `http://arxiv.org/abs/cs.CR/0012023`.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RSA93] PKCS #1: RSA encryption standard. Version 1.5, November 1993. Available from `http://www.rsaisecurity.com/rsalabs/pkcs/`.

[Wil74] Malcolm J. Williamson. Non-secret encryption using a finite field, 1974. Available from `http://www.cesg.gov.uk/publications/index.htm`.

[Wil76] Malcolm J. Williamson. Thoughts on cheaper non-secret encryption, 1976. Available from `http://www.cesg.gov.uk/publications/index.htm`.