The Cost of Cryptography

Yevgeniy Dodis, New York University

November 11, 2013

Cryptography is a science of protecting the privacy and authenticity of sensitive information from unauthorized access. The need for cryptography is clear, and goes back to the ancient times, when kings tried to design "secret codes" to ensure that the messages they exchange with their generals remain secret and also cannot be forged, even if the messenger is captured by the enemy. Over the years, the usage of cryptography has increased dramatically, as it was no longer used exclusively by governments and spy agencies, but became widely deployed by banks, medical institutions, and even average citizens. Still, given the mathematical maturity of the field, there are a lot of situations where using cryptography makes sense, but is not done. This raises the following question: what is the "cost" of using cryptography, and when is this cost lower than the "benefit" that it buys?

This question has evolved a lot over the years, culminating with a very surprising twist in the last several months. But, before getting there, we need to develop some notation, concentrating on the most basic question of encryption. Given a message *m*, called the *plaintext*, we wish to design an *encryption* function *E*, and a *decryption* function *D*, such that: (a) "no information" about *m* is leaked from the *ciphertext c = E(m)*, and; yet (b) the message *m* can be recovered by computing *D(c)*. The classical (and very bad) encryption scheme is the infamous *Caesar cipher*, where the message is encrypted letter-by-letter, with each letter simply "cyclically shifted" by 3 letters. For example, *E(a) = d*, *E(b) = e*, …, so that *E(hello) = khoor*. Of course, the security of this cipher is based on the assumption that the encryption and decryption algorithms remain secret, which is very unrealistic over a long-term use. It has many other problems, but the above concern almost immediately leads us the famous **Kerckhoffs's principle:** "A cryptosystem should be secure even if everything about the system, *except the key*, is public knowledge."

So what is this "key"? It is an extra input *k* to the encryption and decryption algorithm, which is assumed random and **unknown to the attacker**: *c = E(m,k)*, *m = D(c,k)*. Intuitively, without *k* the attacker cannot simply run the decryption algorithm, leaving hope that *c* might indeed hide *m*. For example, one naive way to "fix" the Caesar cipher would be to make the actual shift to be this key *k*, and set *E(m,k) = m + k mod 26*, *D(c,k) = c – k mod 26* (where we identify letters *a…z* with numbers *0…25*). Still, even if this shift *k* is random and secret, there are two obvious problems: (1) there are only 26 possibilities for *k*, so an English message can be easily decrypted by trying all 26 keys (this is called "brute force attack"); (2) the ciphertext leaks a lot of partial information about the message (e.g., same letters of *m* are guaranteed to be encrypted into same letters of *c*). Problem (1) suggests that the key should be long enough to avoid the brute force attack, and problem (2) suggests that we need a formal definition of what it means that "*c* leaks no information about *m*" (since, otherwise, we can never be sure that we did not miss some subtle attack).

These challenges where undertaken in a seminal paper of Claude Shannon in 1949, who defined the notion of *perfect security* for encryption. Intuitively, it states that for any a-priori message distribution *M*, if the message *m* is chosen according to *M* and the key *k* is chosen at random, then the ciphertext *c = E(m,k)* is *independent* of *m*: the a-posteriori distribution on *m* conditioned on any particular value of *c* is still *M*! Shannon's work also defined the first formal version of our "cost of cryptography" question:

Question 1: What is the minimal length of key *k* sufficient to perfectly encrypt an N-bit message *m* (or many messages whose total length is N)?

Shannon's work also gave the answer to this question. The key *k* has to be at least as long as the message *m*. Intuitively, if c leaks no information about m, then by decrypting *c* using all possible keys *k*, we must be able to get every one of 2^N possible messages m. Moreover, this bound was tight, as shown by the *classical one-time pad scheme*. The key *k* is a random N-bit string, and $E(m,k) = m\oplus k$, where $\oplus$ is the bit-wise "exclusive-OR" operator (decryption is the same $D(c,k) = c\oplus k = (m\oplus k)\oplus k = m\oplus(k\oplus k) = m$).

Due to its very strong "perfect security", the one-time pad scheme was extensively used for various spy and military operations, despite the huge inconvenience of having to generate fresh "one-time pads" *k* for every new message *m*. This inconvenience actually comes in several forms. First, the sender and the receiver need to generate a huge amount of key material, if they want to have long secret exchanges. Second, they must remain synchronized throughout encryption and decryption, to ensure that they use the same key *k* for a given message. And, most importantly, they need to ensure that the same pad *k* will *never be reused*. Indeed, one of the most amazing breakthroughs for western counter-intelligence against the former Soviet Union came from the famous VENONA project, where the NSA discovered that some of the top secret transmissions by the Soviets accidentally reused the same one-time pads, which allowed complete or partial decryption of approximately 3000 top secret messages.

These limitations of perfect security meant that cryptography could not be widely deployed early on, as the problems of secret key distribution and coordination made it completely impractical. Fortunately, many cryptographers realized that Shannon's definition might be unnecessarily strong, and more "convenient" cryptography is possible if one assumes that: (a) the attacker is "computationally bounded" (e.g., cannot run for more than 2^{80} steps in any realistic amount of time, even on computers which *will be built in any foreseeable future*); and (b) it is OK that an efficient attacker can "break" the scheme with "negligible" probability (e.g., it is OK if the attacker can decrypt the message with probability 2^{-80}, as such miniscule odds do not pose any realistic threat) . As it turns out, both of the relaxations (a) and (b) are necessary to overcome Shannon's pessimistic result, but, more importantly, they also “seem to be sufficient”. This means that, although we cannot formally prove the “relaxed security” of modern encryption schemes under assumptions (a)-(b), modern cryptography developed a lot of practical and theoretical evidence that it is possible to design “computationally secure” encryption schemes. Namely, schemes where the length of the key is *fixed* (to, say, 128 bits, to prevent brute force attacks), but no “reasonably efficient” attacker can “break” the encryption scheme with “non-negligible probability” (the quotation marks can all be formalized, but this is beyond the scope of this article).

The development of modern encryption schemes --- called *symmetric-encryption* schemes (for the reason explained shortly) --- where the length of the key is fixed and independent of the length of the messages to be encrypted, led to a new variant of the “cost of cryptography” question.

<u>Question 2</u>: Are there symmetric encryption schemes where the running time of the encryption and decryption procedures is comparable to the cost of transmitting the unencrypted messages?

For example, assuming one can stream an unencrypted movie in time T, can one stream the encrypted movie in roughly the same time $T' \approx T$? This question took several decades to be resolved, but with the improved processing speeds, coupled with the design of *very fast* encryption schemes (such as encryption schemes based of the so called Advanced Encryption Standard, AES, standardized in 2001), the current answer is resoundingly positive: symmetric-key encryption comes at almost no noticeable computing cost, even on relatively small devices, such as tablets and smartphones. In fact, in the last decade we started to use a lot of symmetric-key encryption, without even noticing it: all connections to “secure web sites” (those which start with “https” as opposed to “http”) are encrypted and, although it took longer than expected, most of our email is also encrypted (Gmail started to do in in 2010, Hotmail – in 2012, and Yahoo will start doing it in 2014).

Unfortunately, the key length and the speed of symmetric-encryption schemes was only one of the important “cost” factors. The major inconvenience, which still remains, comes from the fact that the sender and the receiver need to agree of the same random key *k*. While this could be feasible if the there are few senders and receivers who could conveniently meet at some place to exchange the keys (e.g., for military or espionage purposes), modern days average users need to be able to securely talk to hundreds of merchants, as well as other users located in different parts of the world. This means that the problem of secret key distribution became the next *major bottleneck* of using cryptography:

<u>Question 3</u>: Can one design convenient key distribution schemes, where people in different parts of the world can easily exchange keys and communicate securely?

This challenge is only partially answered to this day, but the main tool in use is the development of so called *public-key (or asymmetric-key) cryptography*, which was first suggested in the seminal work of Diffie and Hellman in 1976. The key idea is the following. Instead of using the same “symmetric” key *k* for both encryption and decryption, one generates two "asymmetric" keys, *ek* for encryption and *dk* for decryption, and sets *c = E(m,ek)*, *m = D(c,dk)*. The point is that *the encryption key ek can be made public, without sacrificing the privacy of the message m*. Now, if a user Alice wants to communicate with several a-priori unknown people anywhere in the world, she just needs to publish her public key *ek*, and then anybody can encrypt the message to Alice using *ek* (but not knowing *dk*).

The invention of public-key cryptography replaces the problem of secure distribution of *secret*, symmetric keys *k* to the one of *public*, asymmetric keys *ek*, but raises the following question:

<u>Question 4</u>: How efficient are the public-key cryptosystems as compared to symmetric-key cryptosystems?

The answer turns out to be mixed. On the one hand, for encrypting short messages, public-key cryptosystems are *noticeably (perhaps 100 times) slower* than symmetric-key cryptosystems. This means it would be impractical to use public-key encryption if lots of short messages are to be encrypted. On the other hand, the following simple trick shows that, for long messages, public-key encryption is almost as fast as private-key encryption. To encrypt a long message *m* under a public-key scheme (*E,D*), one can first encrypt a fresh, *short* symmetric key *k* under (*E,D*), and then use *k* to encrypt *m* under a *much faster* private-key scheme (*E',D'*). If *m* is long enough, then the "slow" encryption time of short *k* under *ek* will still be small compared to the "fast" encryption of long *m* under *k*. More generally, when lots of (possibly short) messages are to be exchanged between two parties, one can use the same short *k* for some period of time when the parties are communicating, and then erase this *k* when the communication is over (this is why this *k* is called an "ephemeral" key).

Still, despite the huge appeal of public-key cryptosystems, their widespread use is still rather limited. The problem is that the distribution of public keys still requires what is called a *Public-Key Infrastructure* (PKI): a mechanism allowing users to "deposit" their own public keys, and obtain *authentic* public keys of others. In particular, it is very important that the users are assured that the public keys they obtain indeed belong to the "right" users. Indeed, if an attacker can place a rogue public-key into the PKI, and claim that this key belongs to somebody else, then the attacker suddenly gets access to privileged information, by tricking other users to encrypt under the wrong key. This leads to the next challenge of using cryptography in everyday life:

<u>Question 5</u>: How to build a trustworthy and transparent PKI?

Currently, the authentication of public keys is performed by trusted entities called Certificate Authorities (CAs). The job of a CA is two-fold: first, it has to verify the claimed "identity" of the public-key owner (which is usually done using traditional, non-cryptographic means), and second, if satisfied, it has to issue a "certificate" of the form "this public-key *ek* key belongs to this particular user, and is valid for this period of time". The certificate is obtained by another public-key primitive called a *digital signature*, which can be viewed as a dual of public-key encryption: a secret signing key *sk* allows the key owner to digitally sign any message *m* (without hiding *m*), while the corresponding verification key *vk* allows anybody to verify the validity of the above signature, without being able to "forge" a valid signature of any other message *m′*. Thus, a certificate is simply a digital signature of the message *m* = "this public-key *ek* key belongs to this particular user, and is valid for this period of time" under the public-key vk_{CA} of the certificate authority CA.

Fortunately, just like with public-key encryption, we know many efficient digital signature schemes which we strongly believe are secure against any computationally bounded attacker. This means that the design of a good PKI reduces to three non-trivial "trust problems": (a) obtaining correct verification keys vk_{CA} of "trustworthy" CAa; (b) trusting that the CA will not issue valid certificates to "wrong" users (either intentionally or by insufficient identity verification); (c) trusting the CAs will protect the secrecy of their own signing keys sk_{CA} from attackers, as knowing the latter will allow the attacker to issue arbitrary fake certificates, as if coming from the CA itself.

Unfortunately, implementing (a)-(c) correctly turned out to be quite challenging. In practice, problem (a) is solved by having major browsers (Firefox, Chrome and Internet Explorer) physically hardwire the list of trusted CAs (called "root CAs"). However, while the initial list of root CA was quite small, over the years the list has expanded considerably. Moreover, for reasons of scalability, root CA are also allowed to certify several smaller CAs (and so on). In fact, according to the "SSL Observatory" project by the Electronic Forntier Foundation, in early 2013, Mozilla, Microsoft and Google certified about 150 root CAs, which in turn certified about 1500 other CAs as entities whom average users should *unconditionally trust*. Needless to say, most of these entities are completely unknown to average users, and the odds that *all of them* can solve problems (b) and (c) appear alarmingly low. In fact, in the last few years several high-profile scandals have surfaced around various CAs. In 2011, a fraudulent Google certificate was signed under the key of the Dutch CA called DigiNotar, which was used (by still unknown entities) for serious "man-in-the-middle" attacks on Google users in Iran. Around the same time, a Turkish CA TURKTRUST discovered it had "accidentally certified" (according to them) two intermediate CAs in August 2011, including the one used to sign another fake Google certificate.

The above challenges show that building a good PKI is very difficult, making the widespread adoption of cryptography rather slow. They also raise another very interesting point: the successful attacks against CAs did *not* attempt to break the "mathematics" of the actual cryptographic system (e.g., forge a signature or decrypt a ciphertext), but instead proceeded by going *"around"* cryptography. Namely, although the details of these (and other similar) breaches are still surrounded by mystery and speculation, the fake certificate was issued by either hacking into the CA, compromising the CA, or "tricking" the CA into signing a wrong certificate, but *not* by finding a "mathematical breakthrough". More generally, although several cryptographic schemes (such as a well-known "Merkle–Hellman Knapsack cryptosystem") proposed in the early days of cryptography were quickly broken, none of the major (symmetric- and public-key) schemes used today appear to have mathematical weaknesses which can lead to realistic exploits (provided that the key size is not chosen low enough to allow for a brute force attack).

In contrast, several high-profile compromises of cryptographic systems reported in the last decade managed to do so *without breaking the cryptography surrounding the system*! For example, several recent studies found catastrophic weaknesses in many Random Number Generation (RNG) procedures used for generating public-secret key pairs of various popular cryptosystems. Namely, instead of using high-quality, *fresh* randomness (which, as we recall, is critical for the security of the system), many implementations used *correlated* randomness for generating secret keys of different users, which led to a complete recovery of these secret keys from the corresponding public keys.

In a different vein, another hugely effective line of attacks comes from the so called physical, or *side-channel*, attacks. In these attacks, pioneered by Paul Kocher in 1996, the intruder gets access to some (seemingly benign) physical information surrounding a normal run of the given cryptographic system, such as timing information, power consumption, acoustic data, cache misses, etc. Quite remarkably, by carefully comparing this "side information" with the specification of the given cryptographic algorithms (which, recall, is possible due to the Kerckhoffs's principle), the attacker can often completely recover the secret key! For example, Kocher's original timing attack used the fact that the running time of a popular public-key cryptosystem called RSA is highly correlated with the number of 1s in the bit decomposition of its secret key. Moreover, by additionally manipulating the (public) inputs to the RSA algorithm, repeated executions with the same key can be used to perform certain statistical correlation analysis to recover the key completely.

However, perhaps most shockingly, the recent revelations by the former NSA contractor Edward Snowden earlier this year confirmed (among many other things) that **the most successful large scale attacks against cryptography - by a wide margin – do not involve either mathematical breakthroughs, or unexpected RNG weaknesses, or sophisticated side-channel attacks.** Instead, these attacks, including the bulk of the mass surveillance program allegedly run by the NSA against American citizens, got access to privileged and confidential information *in the cleartext form*: either before encryption, or after decryption, or simply sitting in unencrypted form in places which were *assumed to be secure*. While many details of this scandal are still being sorted out, it raises the following surprising twist on our main question which we promised at the beginning of this article:

<u>Question 6</u>: **given most real-world cryptosystems get broken without breaking the "mathematics of cryptography", is cryptography a "waste"?** Put differently, given the costs and the challenges involved in using cryptography, are these costs justified in light of the current, largely "non-cryptographic" attacks?

Fortunately, despite the provocative nature of the question, the answer is pretty non-controversial. Clearly, all the non-cryptographic attacks above require either implementation mistakes (which we can hopefully avoid in the future), or pretty non-trivial resources from the attacker. E.g., the side-channel attacks require pretty intimate access to the secret key operations, while the (alleged) mass surveillance by the NSA requires enormous financial, technical or political resources to be run effectively. In contrast, without cryptography much simpler and cheaper attacks would be possible. To put it differently, **cryptography is not a bullet-proof vest, but only *one of several necessary components* required to protect the privacy and the authenticity of sensitive information.**

If anything, cryptographers should take pride in the fact that they did their "part of the puzzle" too well, and most successful security exploits tend to go *around* cryptography, and not *through* cryptography. Still, this raises our final take on whether cryptography is "wasteful", this time focusing on the *future* of the field; more specifically,

<u>Question 7</u>: given most real-world cryptosystems get broken without breaking the "mathematics of cryptography", should cryptographers be satisfied, and is their still a need for new cryptographic research?

Fortunately (or, unfortunately, depending on one's point of view), the need for cryptographic research is *even higher* than what it was in the past. First, one can design new cryptographic systems which are provably resilient so at least some of the above mentioned "non-cryptographic" attacks. For example, a lot of recent research was devoted to the question of building so called "*leakage-resilient cryptosystems*", which are mathematically proven to resist large classes of well-defined, realistic side-channel attacks. Similarly, lots of research is done to make cryptosystems secure even in light of weak randomness, or, alternatively, to design better random number generators for cryptographic use.

Second, the vast amount of sensitive data stored and generated in digital form created completely new cryptographic challenges, which are much more advanced than the basic question of designing good encryption and authentication mechanisms. For example, how can we securely store our data "in the cloud"? How to access this data quickly, without violating the privacy of the data? How can we ensure that the data is not lost, without excessively replicating this data? How can we perform computations on the data in the cloud, without retrieving all the data, or violating the privacy of the data? How can we delegate partial or temporary access to our data, but have the ability to revoke this access?

These and many other exciting challenges have led to a variety of amazing cryptographic tools developed in recent years. For example, cryptographers developed tools for meaningfully *computing on encrypted data*, without violating the privacy of this data! Hence, in principle it is possible for a user to encrypt its search query Q and send the *encrypted* query to Google, who can then return a correct (but still encrypted!) answer to the user, without having any idea about the actual search query Q! (As a warning, the current encryption schemes with this incredible capability, called *fully-homomorphic*, are still too inefficient to be used in everyday life, but this is bound to change, and hopefully soon.)

The above example of computing on encrypted data also shows that the abundance of digital data and the emergence of cloud computing not only bring new challenges to overcome, but also offer amazing cryptographic capabilities which did not appear realistic only a few years ago. For another example, while the users can already utilize various “cloud computing services”, such as Amazon’s EC2 and Microsoft’s Azure, to perform various computationally expensive tasks, new cryptographic schemes were recently developed where the users can also check the correctness of these computations in a way requiring *much fewer resources* than doing the actual computation.

Many other exciting cryptographic techniques are being developed, such as electronic cash, anonymous credentials, secure multi-party computation, electronic voting and privacy-preserving data mining, to name a few. While some of these schemes are either impractical at the moment, or appear way ahead of the current technology, others are slowly being deployed in our lives. For example, the *Bitcoin* system is a distributed, peer-to-peer digital currency which received a lot of publicity, and is being increasingly used as payment for various products and services, due to its low transaction fees as compared to the more traditional payment mechanisms.

To sum up, although the deployment of cryptography still faces many challenges, the “cost-to-benefit ratio” of using cryptography is decreasing at a very fast pace, and the future of the field appears quite bright. On the other hand, although we have come a long way since the VENONA project and single-use keys, we continue to live the lesson that *security should not just be strong, but also seamless*.