The ISC Int'l Journal of Information Security

July 2011, Volume 3, Number 2 (pp. 69–76)
http://www.isecure-journal.org

Invited Paper

Constructing Cryptographic Definitions

Philip Rogaway [1]
[1] *Department of Computer Science, University of California, Davis, California, USA*

ARTICLE INFO.

Article history:
Received: 10 September 2011
Revised: 12 March 2012
Accepted: 13 March 2012
Published Online: 19 May 2012

ABSTRACT

This paper mirrors an invited talk to ISCISC 2011. It is not a conventional paper so much as an essay summarizing thoughts on a little-talked-about subject. My goal is to intermix some introspection about definitions with examples of them, these examples drawn mostly from cryptography. Underpinning our discussion are two themes. The first is that *definitions are constructed*. They are invented by man, not unearthed from the maws of scientific reality. The second theme is that *definitions matter*. They have been instrumental in changing the character of modern cryptography, and, I suspect, have the potential to change the character of other fields as well.

1 Introduction

Let me first try to clarify what I mean when I speak of a *definition*. First, a definition here is something that embodies an important concept in a field. If I say "let $m = \lceil \sqrt{x} \rceil$", that's not the kind of definition I have in mind. Second, I insist that definitions be mathematically rigorous. If I say "a message authentication code allows the recipient of a message to verify the claimed identity of its sender," I've given a description, not a definition.

What do I mean when I say that definitions are *constructed*? I am using the term here as it is used in sociology. When we say that some thing, C, is constructed, or socially constructed, we are emphasizing that C need not be the way that it currently is. It is not inevitable. Instead, C is contingent on social forces, or it springs from aspects of our disciplinary culture. I mean, in short, that C was invented.

Those from an engineering background might implicitly assume that all of cryptography is constructed. But there is an alternative viewpoint, and one that, I suspect, is quite popular among those who work on the more rigorous side of cryptography. This alternative viewpoint is what might be called *scientific realism*. Here we say that C is the way it is because that is the nature or mathematical or physical reality. In order to have a successful theory involving C, it pretty much has to be as it is now. If C is shaped by the disciplinary culture, this happens in a superficial way. In short, C is discovered, either through reasoning or the exercise of the scientific methodology.

Nobody would contest the claim that a concrete protocol or primitive, something like AES or TLS, is constructed. What is at issue is whether or not our basic notions in cryptography—things like a one-way function or a secure encryption scheme—whether these are invented or discovered. Constructionism or scientific realism.

The thesis here is that *all* of cryptography's notions are highly constructed. As a consequence, the field can move in very different ways from the way that it has moved.

Email address: rogaway@cs.ucdavis.edu (P. Rogaway).
ISSN: 2008-2045

2 NP-Completeness

While this essay is about definitions in *cryptography*, I would like to begin with a definitional example from outside this field. I do this to ground our discussion in the familiar; hopefully most readers have seen a definition of NP-completeness, due, independently, to S. Cook and L. Levin, circa 1971. We speak of a language L being *NP-complete* if two conditions hold: L is in the set we call NP; and for any language A in NP the language A polynomial-time reduces to L. Said differently, L is a hardest language in NP.

You've just seen an example of a definition—indeed a *wonderful* definition. I haven't defined all of its constituent parts—I didn't define NP or tell you what it means for one language to polynomial-time reduce to another. Let me skip over that.

Why do I say that this is a wonderful definition? There are a couple different senses in which I could justify such a claim. One is an *a priori* assessment of the definition's value. We could say that the notion of NP-completeness is *good* because it is particularly elegant, simple, or potentially useful. We could argue that it captures strong intuition, or has various nice properties. You could make this kind of argument.

An alternative is to take an *a posteriori* view. You would say that a definition is *good* if it comes to spawn lots of work that you hold to be interesting. It is in this second sense that a definition like NP-completeness really shines. Back in 1995, C. Papadimitriou did a literature survey and found that, already, there were more than 6000 papers per year having the term *NP-complete*—more than the number of papers with the word *compiler*, *database*, or *operating system*. Today, more than five million web pages contain the term *NP-complete*, and more than 137,000 Google-Scholar articles.

It is my view that this after-the-fact evaluation of a definition is the preferred way to understand how good a job a definition has done. Definitions are created to benefit some community, so the insightful inventor of a definition has foreseen what it is that this community needs, and he has provided a foundation to help it move in that direction. That is what good definitions in cryptography manage to do.

3 Provable Security

Definitions are not very old in cryptography. They emerged rather suddenly, around 1982, in a paper of S. Goldwasser and S. Micali. Before this, the "classical" approach in cryptography consisted of recognizing some problem, devising some scheme that aimed to solve it, and waiting to see if any interesting attacks emerged. When they did, one would revise the scheme and try again. Goldwasser and Micali suggested a way to do better. In the framework they put forward, one does not begin with a protocol; one begins with a definition. Once it has been carefully laid out, *then* one devises a protocol. Ideally, you would now prove that your protocol satisfies the definition. In practice, we usually give proofs that take the form of a *reduction*. The proof establishes that some protocol Π meets its definition D as long as some *other* protocol π meets *its* definition d. If you're confident that π is good in sense of definition d, you'll have to believe that Π is good in the sense of definition D.

The above idea of *provable security* utterly transformed the field. Cryptography went from being an *ad hoc* set of techniques to a scientifically rich area well connected to complexity theory, mathematics, and computer security. Nowadays, I would say that about half of all work in cryptography falls within the provable-security tradition. While initially there was minimal impact of this line of work on cryptographic practice, this has changed. Provable security has now come to interact synergistically with the classical approach to doing cryptography and has given rise to many practical and high-assurance techniques.

From what I've laid out, you might infer that the overarching purpose of definitions is to enable theorems and proofs. And it is of course true that definitions are essential for these activities. But I think that definitions would be important in cryptography even if we *never* used them to give a proof. First, definitions can lead to *attacks*. (That attacks can lead to definitions is all the more clear.) Once you carefully define the goal you are after, you can quite often use that understanding to see how to break a protocol that was supposed to meet its formerly-undefined aim. Second, definitions are essential for *productive discourse*. In cryptography, it seems like a lot that is said doesn't make a whole lot of sense. Only when the definitions are clearly articulated can you really know what it is you're talking about. Finally, definitions seem to be essential for fostering our ability to think and understand. Thinking in complex domains involves building up abstraction boundaries, and, in many fields, these are embodied by definitions.

4 Pseudorandom Generators

So far we have given a single example of a definition (NP-completeness). My remaining examples will all be cryptographic. I will start with the notion of a *pseudorandom generator*. The informal goal here is to create bits that look random (uniformly distributed),

even if they are not.

In trying to give a definition for this goal it is important *not* to think in terms of anticipated solutions. Instead, one tries to understand what it is that the question *means*, what it *is* to generate random-looking bits.

The first thing one might imagine is that one can look at two strings and *see* that one looks more random than the other. For example, most would claim that the string

$$Y = 00101110100001001110101101111101101110$$

appears to be more random than the string

$$Z = 00000000000000000000000000000000000000.$$

And there *is* such an approach for dealing with randomness, Kolmogorov complexity, that dates to 1960's. But this is not the approach that has had much cryptographic significance. Instead, what Blum and Micali, and Yao suggested, in the early 1980's, was that pseudorandomness is a property not of a string but of a probability distribution.

We begin with *syntax*. The object we are interested in studying, a *pseudorandom generator* (PRG), is a function that maps a "short seed"—a binary string—into a longer string. In other words, a PRG is a map $G\colon \{0,1\}^n \to \{0,1\}^N$ where $n < N$ are constants.

We want to measure the "quality" of a PRG. If you realize a PRG G by $G(S) = S \,\|\, S$, for example, this doesn't seem like a *good* PRG, even though it does comply with our syntax.

Let's suppose $n = 100$ and $N = 200$—we aim to stretch 100-bit strings to 200-bit ones. If you take a string $S \in \{0,1\}^{100}$ that is truly random, and compute $Y = G(S) \in \{0,1\}^{200}$, you'll now have an induced distribution on 200-bit outputs. There is probability mass on at most 2^{100} points of our size-2^{200} space of possible outputs. So it's actually a very sparse subset of the 200-bit strings that could ever arise as pseudorandom outputs. The idea for our definition of a PRG's quality is to say that it doesn't actually matter. If you give an adversary a string that is formed by taking a random 100-bit string S and applying G, or if, instead, you give the adversary a truly random 200-bit string, our poor adversary won't be able to tell the difference.

More precisely, an adversary A is imagined to possess one of two kinds of oracles. One possibility: the adversary hits a button and, in response, a random 100-bit string S is selected and the adversary is presented the 200-bit $G(S)$. She can hit the button as many times as she likes, each time a random S being chosen afresh. Call this "first" world. Alternatively, the adversary hits the button and, in response, gets a random 200-bit string. She can again hit the button as many times as she likes. Call this the "second" world. We understand our PRG G as *good* if for any *reasonable* adversary A, its ability to distinguish if it is in the first world or in the second world is *small*. To make this precise, we associate to A and G a real number $\mathbf{Adv}^{\mathrm{prg}}_G(A)$ that captures the adversary's ability to distinguish its two possible worlds:

$$\mathbf{Adv}^{\mathrm{prg}}_G(A) = \Pr[A^{G(\$)} \Rightarrow 1] - \Pr[A^{\$} \Rightarrow 1]$$

This is the probability that A outputs "1" if we answer its button-pressing with pseudorandom bits minus the probability that it outputs "1" if we answer its button-pressing with truly random bits. Advantage 0 means the adversary does terribly. Advantage near 1 means the adversary does great.

We have now given a cryptographic definition. The definition is a way of associating to a cryptographic object (the PRG G) and an adversary (the algorithm A) a real number, the number telling us how well the adversary is doing in attacking the object's aim.

5 Asymptotic vs. Concrete Security

We still haven't made a firm distinction between a *good* and a *not good* PRG. The traditional approach to drawing this distinction is to say that G is *secure* if it's computable in polynomial time (in n) and for every probabilistic polynomial time adversary A, the advantage $\mathbf{Adv}^{\mathrm{prg}}_G(A)$ that A gets in attacking G is a *negligible* function. The technical definition for the last term: $\varepsilon(n)$ is negligible if it vanishes faster than the inverse of any polynomial: for every $c > 0$ there exists an N_c such that $\varepsilon(n) < n^{-c}$ for all $n \geq N_c$.

The above *asymptotic approach* lets us draw a rigorous distinction between secure and not secure PRGs. We have, in the process, quietly changed our notion of a PRG: n and N are no longer constants but, rather, the PRG should operate on strings of any length n, returning outputs of length $N(n) > n$. The value n is now called the *security parameter*.

There's an alternative approach for defining security. It says: don't try to make binary distinctions. Once we have said how to associate an advantage $\mathbf{Adv}^{\mathrm{prg}}_G(A)$ to each A and G, we *have* defined security. At this point one can already state theorems relating the advantage that a first adversary can obtain at attacking a first goal to the advantage that a second adversary can obtain in attacking a second goal. This is the *concrete-security* approach that M. Bellare and I helped popularize.

I'd like to ask if this distinction between asymptotic and concrete security is important, if it's a *significant* difference. The asymptotic approach came first, and somehow it took a long time until people started to supplement this with concrete security.

One answer you can reasonably give is to say that the difference between asymptotic and concrete security is not at all significant because, first, asymptotic definitions and theorems can almost always be converted into concrete-security ones. The essential ideas of a definition, theorem, or proof almost always transcend this concrete vs. asymptotic distinction. In general, I think it's fair to say that good definitions, in cryptography and beyond, are quite *robust*, in the sense that diverse elaborations of definitional choices leave an intact definitional core.

But you can also make the case that the asymptotic vs. concrete definitional choice is quite significant. In particular, the character of cryptography was profoundly influenced by the early choice of an asymptotic approach. Because asymptotic analysis hides "low-level" efficiency matters, treating all polynomials as equivalent, and all negligible functions as equivalent, people tended to focus on broad, abstract relations among cryptographic goals. There was little interest in efficiency. In addition, the asymptotic approach went hand-in-hand with a preference for public-key (or "asymmetric") cryptography, the lovely idea put forward by W. Diffie and M. Hellman. Here each party generates a public key and keeps secret a corresponding secret key. Shared-key (or "symmetric") cryptography tended to be ignored by theorists, or even denigrated. We would suggest that one reason for this preferential interest in asymmetric cryptography among theorists is the need for a security parameter in asymptotic treatments. Security parameters are natively present in most public-key schemes, but are usually absent in symmetric schemes. Since practitioners have always been interested in efficiency and in symmetric constructions, I would maintain that the asymptotic approach to cryptography helped give rise to a social phenomenon wherein practitioners tended to ignore theorists, and theorists tended to ignore practitioners. Each group came to see the work of the other as largely irrelevant.

The character of cryptography changed in consort with the popularization of concrete security. Theorem statements became more precise, and with that one started to attend to lower-level relationships between the security of schemes. New questions became visible, things that you simply do not see if you describe everything in terms of asymptotic security. Symmetric cryptography joined that ranks of topics having legitimate scientific credentials.

I would conclude from this example that specific definitional choices dramatically affect the way a theory develops, and what it is good for. Definitional choices impact the types of questions that will be asked, and the types of questions that will be rendered invisible. I would also conclude that definitions arise within a particular disciplinary culture. It makes sense, in retrospect, that definitions in cryptography would initially have been asymptotic, because the founders of the field were coming from a community that had recently mastered the idea of NP-completeness, and other complexity classes, a tradition that was already steeped in reductions, polynomiality, and asymptotics. Making the simplest transition from this world to cryptography meant that we were going to create cryptographic foundations that were asymptotic and would emphasize the kind of high-level questions that complexity theory had also come to focus on.

Definitional choices do more than reflect our disciplinary culture and sensibilities. Once those choices have been made, they effectively reinforce that culture and those sensibilities, distancing us from other and outside concerns. It is a feedback phenomenon. As Marshall McLuhan has colorfully explained, we shape our tools, and then our tools shape us.

6 Blockciphers

Let's move on to another example, blockciphers. These are a basic building block of symmetric cryptography, and I suspect that everyone reading this essay knows some example blockciphers, like DES and AES. The question I want to ask here is what a blockcipher *is*. In answer, a blockcipher is a function that takes in a key K from some finite set $\mathcal{K}$ of possible keys, and it takes in an n-bit plaintext block for some constant $n > 1$. It produces a corresponding ciphertext block, again of length n. We require that each $E_K(\cdot) = E(K, \cdot)$ be a permutation, a one-to-one and onto function.

The above is the *syntax* of a blockcipher; as with our treatment of PRGs, I've begun without specifying anything about security. There are lots of approaches to trying to define security. For example, you might create a definition out of the intuition that a blockcipher is good if it is hard to recover the key from witnessing the input/output behavior of the blockcipher. Or you might try to capture the property that it is difficult to recover the plaintext given the ciphertext. Or you might focus on the unpredictability of ciphertexts for unknown plaintexts. All of these notions *can* be built up into definitions—we can come up with a rigorous **Adv**-notion for each. But none of these ideas really work to give us a convenient-to-use cryptographic primitive. The winning approach

is to capture that a blockcipher should behave like a random permutation.

Here I will sketch the *pseudorandom permutation* (PRP) notion for blockcipher security. As in the pseudorandom generator setting, we imagine an adversary dropped into one of two possible worlds. In the first of these worlds, the adversary, A, is given access to a box that computes the blockcipher E for a randomly chosen key K. At the beginning of the game, a random key K is selected from the key space $\mathcal{K}$ and you give the adversary blackbox access to the function $E_K(\cdot)$. The adversary can query whatever plaintext blocks it likes, getting, in response to each $X \in \{0,1\}^n$, the output $Y = E_K(X)$. Each query the adversary asks can be based on the prior outputs it has learned. In the *second* world, the adversary A, in response to each query X, gets the image of a random permutation π (again from n bits to n bits) applied to X. In other words, to each new query $X \in \{0,1\}^n$ we return a new, uniformly chosen $Y \in \{0,1\}^n$. If any query is repeated, we answer as we did before. We measure the adversary's advantage by

$$\mathbf{Adv}^{\mathrm{prp}}_E(A) = \Pr[A^{E_K} \Rightarrow 1] - \Pr[A^{\pi} \Rightarrow 1].$$

This is the probability that the adversary outputs 1 when we drop it into the first world, minus the probability that it outputs 1 when we drop it into the second world.

This is the second cryptographic definition we've described. Informally, blockcipher E is *secure* as long as for every *reasonable* adversary A—adversaries that don't spend *too* much time computing, have description size that's not *too* big, and don't ask *too* many queries—the advantage $\mathbf{Adv}^{\mathrm{prp}}_E(A)$ is *small*—it's a number close to zero.

The PRP definition has been enormously productive. Nowadays, when we speak of a blockcipher, theorists usually *mean* something that does well with respect to the definition just described. Even cryptanalysts have come to accept these notions, no longer viewing key recovery as the one and only property to violate to have a convincing attack.

I would draw a few conclusions from our account of blockciphers. First, as with PRGs, we separated the syntax of the object from its security notion. I believe that this is always the right thing to do. Second, simple, pessimistic definitions—meaning that they give the adversary credit quite generously—are often better choices than more complex and faithful ones. Our definition was only a thought experiment for defining security; in making a definition like this we are not trying to faithfully capture an adversary's capabilities in some usage environment; we are seeking a simple definition that pessimistically measures the worth of the scheme. Finally, I would say that definitions can, in fact, be *wrong*. The examples I gave earlier of definitional routes not taken are wrong in the sense that they do not give rise to nearly as useful a starting point.

7 Symmetric Encryption

Next I would like to look at what a *symmetric encryption* scheme is. Symmetric, or *shared key*, encryption is the well-known problem where Alice and Bob want to send messages to each other protected by a shared key, K. When we formalize what an encryption scheme is, the approach, going back to Goldwasser and Micali (1982) and then adapted to the symmetric setting by Bellare, Desai, Jokipii, and me (1997), is this. The encryption algorithm takes in a key K and a plaintext M. It produces a ciphertext C. The encryption algorithm may be probabilistic—it can exploit internal "coins" (randomness) if it so wishes. Correspondingly, the ciphertext may be longer than the plaintext. Of course there should be a corresponding decryption algorithm. It takes in the key and ciphertext and produces a plaintext. Decryption must reverse encryption: $\mathcal{D}_K(C)$ must be M whenever $C \leftarrow \mathcal{E}_K(M)$. Our security notion captures an adversary's inability to distinguish the encryptions of equal-length strings.

The question I would like to ask is whether or not it was *necessary* to formalize symmetric encryption in roughly this way. The thesis I expressed in the Introduction would suggest an answer of *no*. But I can tell you that, when I was working on this problem in the late 1990's, as I saw it then, there really was only one reasonable approach. We had already learned, from Goldwasser and Micali, what was the "right" way for defining public-key encryption. What was needed now a thoughtful adaptation of this notion to the shared-key setting.

I realize now that there are a variety of ways to go. Here's an alternative I now favor—it's usually called *authenticated encryption with associated data* (AEAD). The long name conceals that this is another way to formalize what an encryption scheme ought to be and do. Again focusing on the syntax, an encryption algorithm will now be understood to take in a key K and a message M, but, also, an *initialization vector*, IV, and a *header*, A. From these four inputs the encryption algorithm will produce the ciphertext. It will do so deterministically—no coins allowed. We may assume this time that the length of the ciphertext is the length of the plaintext. As before, there must be a corresponding decryption algorithm. It takes in the

key, the IV, the header, and the ciphertext, and it produces the plaintext or else a distinguished symbol $\perp$, which is used to indicate that the provided ciphertext does not correspond to a valid plaintext. The security notion captures an adversary's inability to distinguish the encryptions of equal-length strings and, also, its inability to produce a new ciphertext having a valid associated plaintext.

The two notions I've sketched are very different views about what an encryption scheme is. It is the second approach that leads, I believe, to mechanisms that are easier to correctly use. First, we do not have to ask our encryption algorithm to generate good random bits; in fact, we forbid them from generating any random bits. The source of "newness" for each message is embodied by the IV. Second, the provisioning of authenticity makes for a scheme that is easier to correctly use. There is a long history of protocol designers implicitly assuming more of their encryption schemes than what the constructions actually provide. Third, in the absence of an explicit header, one could not do something as simple as authenticate the source address in a networking packet. The predictable consequence is to turn users of encryption schemes, those designing networking protocols, into unwitting designers of cryptographic schemes.

It is only since 2004 that we have shared-key encryption schemes architected to the AEAD abstraction boundary. Two of these schemes—CCM and GCM—were quickly standardized by the U.S. National Institute of Standards and Technology (NIST), and others. These modes have already eclipsed traditional modes of operation like CBC as the preferred way to encrypt in higher-level protocols. CCM is the method by which one nowadays encrypts in WiFi networks, while GCM is one of the permitted methods for IPSec.

I would again like to draw some conclusions. First, we have evidenced that questions utterly basic to a field—like the question "what is symmetric encryption?" for cryptography—are highly constructed. The classical definition is as it is because of inessential choices made when the community addressed its initial challenges. Second, I would emphasize that definitions are not written in stone. They emerge, change, and die out far more often than people imagine. They are part of a dialectic within a community. Third, I would conclude that how we define something—simple things like how many arguments get fed into an encryption scheme—can have a profound effect on how useful that object will be.

Smart people can mess up when they don't understand the underlying definition. In 2001, a number of authors put forward fast authenticated-encryption schemes: Jutla; Gligor and Donescu; and myself, Bellare, Black, and Krovetz. The U.S. National Security Agency (NSA) then put out their own proposal for authenticated encryption, which they called "Dual Counter Mode." But I myself broke the proposal within a couple of hours. Others quickly broke it, too. I am not skilled at attacking things, but I understood the definition of what an authenticated-encryption scheme was supposed to do. The folks at the NSA who designed the mode must not have.

I would conclude, finally, that practice that has not yet met theory is an excellent place to be crafting definitions. The definition for AEAD might have emerged ten or even twenty year earlier if theorists had simply reverse-engineered what practitioners were already trying to do. Many theorists seem to believe that theory invariably precedes practice. My own experience suggests that the converse holds at least as often—that practice routinely leads theory, and that it can take a long time for the theory to catch up.

8 Collision-Resistant Hashing

My next example is a collision-resistant hash function. Such an object H takes in a string of arbitrary length and gives a *message digest* of, say, 160 bits: $H\colon \{0,1\}^* \to \{0,1\}^{160}$. Examples include MD5 and SHA-1.

The first property that people speak of in trying to understand what one of these functions is supposed to do is *collision resistance*. You will see "definitions" in the literature of the sort:

> H is *collision resistant* if it is computationally infeasible to find distinct strings X and X' such that $H(X) = H(X')$.

We know that there are lots of such *collisions*—if $H\colon \{0,1\}^* \to \{0,1\}^{160}$ there will already be numerous pairs of 161-bit strings that hash to the same value. The difficulty is in finding such a collision.

To formalize the security goal we can define $\mathbf{Adv}^{\mathrm{col}}_H(A)$ as the probability that adversary A outputs distinct X and X' such that $H(X) = H(X')$. We compare $\mathbf{Adv}^{\mathrm{col}}_H(A)$ to the computational resources used by A. Informally, we regard H as secure if every "reasonable" adversary A gets "small" advantage measure $\mathbf{Adv}^{\mathrm{col}}_H(A)$.

The problem with the above is that it is, well, kind of bogus. No matter what H may be, there will *always* be an efficient algorithm A that outputs a collision for it—namely, the efficient algorithm that *knows* a collision X, X' for H and outputs it.

I will say it again. Specify any hash function $H\colon \{0,1\}^* \to \{0,1\}^{160}$. There will *always* exist an algo-

rithm A that prints out H-colliding 161-bit strings X, X'. The algorithm is as efficient as can be—just a couple lines of code—and it gets advantage 1.

At some level, the above reasoning is clearly specious: of course the collision-printing algorithm *exists*; the difficulty is our inability to explicitly specify it. So we can try it again, saying that a hash function H is collision-resistant if there is no *person*—no living human being—who can write a collision down.

But the above should seem even more silly: if we are trying to come up with a mathematically rigorous treatment of hash functions, certainly you can't base it on what human beings do or do not know. No definitions in mathematics have such a character.

What is the solution to this foundational dilemma? The way that theorists have usually addressed this issue is to say that a cryptographic hash function oughtn't have a signature $H\colon \{0,1\}^* \to \{0,1\}^n$. Instead, we should consider a family of hash functions, our hash-function family having signature $H\colon \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$. Each key $K \in \mathcal{K}$ names a hash function $H_K(\cdot)$ from the family. The user of the hash-function family selects a K and publishes it. The foundational problem vanishes because we will demand the inexistence of an efficient algorithm that, given K, finds collisions for $H_K(\cdot)$. While, true, there will be an efficient collision-finding adversary for each K, there might not be an efficient collision-finding adversary that works for a random K.

The problem with the above move is its fundamental lack of fidelity with respect to modeling real-world hash functions. Objects like SHA-1 were not described as having a key, and reinterpreting them as elements of a hash-function family is inherently a stretch.

In a paper a few years ago (2006), I pointed out that this entire dilemma is all a bit of a misunderstanding. There was never a need to key hash functions to have a sensible security notion. All you have to do is to change the way that you state your theorems. We won't write an "existential" claim like

> If there is an effective algorithm A for attacking the H-using protocol Π then there is an effective algorithm C for finding a collision in H.

We can't state meaningful theorems in this way because the conclusion is always true. Instead, we will make an "explicit-reduction" claim like

> There is an (explicitly given) algorithm R such that when A does well at attacking the H-using protocol Π then $C = R^A$ does well at finding a collision in H.

Now, one's ability to break Π does imply one's ability to break H, exactly what we want.

From where did this entire confusion arise? The first rigorous paper on cryptographic hash functions, by I. Damgård (1987), already explained that

> Instead of considering just one hash function, we [must] consider families of them, in order to make a complexity-theoretic treatment possible.

When I read this statement as a graduate student, I took the claim to be true, even incontrovertible. Years later, I can look back and identify some of the implicit assumptions that made this seem so. First, the statement implicitly assumes an asymptotic treatment of our goals. Even more, it assumes that we want our security notions to hold for *non-uniform* adversaries—those that can have some security-parameter-dependent advice. Finally, we assume existential-format theorem statements. When you abandon this set of assumptions, moving to hash-function families no longer seems so right.

I conclude from all of this that implicit and unrecognized assumptions can determine a good deal of what we think and do. Deeply embedded disciplinary assumptions can come to assume almost doctrinal unassailability. The assumptions can effectively vanish from our view. In the words of L. Fleck (1935/1979), "Once a structurally complete and closed system of opinions consisting of many details and relations has been formed, it offers enduring resistance to anything that contradicts it. ... What does not fit into the system remains unseen." It is not that we see alternatives to how we are doing things, and, carefully considering, reject these possibilities. It is, more, that we never see the alternatives—even things that, later, seem obvious and compelling.

9 Zero-Knowledge

The beautiful idea of *zero-knowledge proofs* was motivated by simple protocols like the one I'll describe right now. A *prover* would like to convince a *verifier* that a pair of graphs G_0 and G_1 are *isomorphic*. (Remember that two graphs are isomorphic if they are the same up to the naming of their vertices.) In order to show this the prover could exhibit the isomorphism: he could provide a permutation that serves to map the vertices of the first graph into the vertices of the second graph, and the verifier would check that the proposed isomorphism really does preserve adjacency. The "problem" with this proof, from a cryptographic standpoint, is that it reveals everything; the verifier now knows the isomorphism. The question I'd like to ask is if you can prove that two graphs are isomorphic without revealing the isomorphism—indeed without revealing anything beyond that fact that the graphs *are* isomorphic.

The problem and its solution, invented by S. Goldwasser, S. Micali, and C. Rackoff (GMR) (1985), begins by having the prover select a random isomorphic copy of the graph G_0. A random isomorphic copy of G_0 is also a random isomorphic copy of G_1 (isomorphism of graphs is an equivalence relation). The prover sends his graph H to the verifier. The verifier now flips a coin $b \in \{0,1\}$ and challenges the prover to demonstrate that G_b actually is isomorphic to H. The prover obliges by exhibiting the requested isomorphism, which the verifier checks.

I claim that when the verifier interacts with the prover I've described, he really does get some evidence that G_0 is isomorphic to G_1. If the two graphs are isomorphic, the prover will be able to provide the specified evidence. But if the two graphs are *not* isomorphic, then H can be isomorphic to at most one of the two graphs. The verifier, who chose b at random, has at least a 50% chance of catching the prover in his lie. If the prover and verifier repeat this game 100 times and then the prover will be able to trick the verifier with probability at most 2^{-100}. For all practical purposes, this number is zero.

Intuitively, the prover manages to establish that G_0 and G_1 are isomorphic without revealing anything about the isomorphism. GMR make this formal with the idea of a *simulator*. The simulator is used to concretize the intuition that the prover leaks zero information in an interaction if that which a verifier obtains is nothing but a sample from a distribution that the verifier itself could generate.

The definition of zero knowledge, and the ideas behind it, have been profoundly influential. More than 17,000 Google Scholar articles, and 149,000 web pages, refer to zero-knowledge proofs. And yet I think it's fair to say that zero-knowledge, at least in the traditional form that I've described, has had little impact on cryptographic practice. It somehow hasn't mattered. I conclude that if a notion is elegant enough, real-world applications may not be needed. I would also comment that *names* can be important. When you have a good notion, and you have a good name for it as well, that is an unbeatable package. The phrase *zero knowledge* manages to capture, in just these two words, a wonderful paradox: how can something be *knowledge* if it is also *zero*? The name itself already inspires the imagination. Good definitions excite the imagination and aspirations of a community.

10 Conclusion

People have come to think of cryptography as a field that's all about schemes, attacks, and proofs. And all of these things *are* vital aspects of cryptography. And yet, missing from this picture are *definitions*. Less well noticed, they are, just the same, at least as important for determining the character of the field.

Acknowledgments

I would like to cordially thank Rasool Jalili, Mohsen Kahani, and everyone else involved in inviting my participation at ISCISC 2011. I would like also to thank the anonymous referee who provided careful comments on this manuscript. Publication timing constraints have made it infeasible to implement all of the referee's insightful comments.

This work was supported by NSF CNS 0904380. Many thanks to the NSF for their continuing support of my research.

Further Reading

[1] Mihir Bellare. Practice-Oriented Provable Security. *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, July 1998*. Lecture Notes in Computer Science, vol. 1561, Springer, pp. 1–15, 1999.

[2] I. Damgård. A "Proof-Reading" of Some Issues in Cryptography. *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007*. Lecture Notes in Computer Science, vol. 4596, Springer, pp. 2–11, 2007.

[3] O. Goldreich. Foundations of Cryptography—A Primer. *Foundations and Trends in Theoretical Computer Science*, vol. 1, no. 1, now publishers, pp. 1–116, 2005.

Philip Rogaway is a professor in the Department of Computer Science at the University of California, Davis, USA. He has also been a frequent visitor to Chiang Mai University, Thailand. His research is in cryptography. He did his Ph.D. at MITs Theory of Computation group (1991), worked at IBM for some time, and then came to UCD (1994). His research has focused on obtaining provably-good solutions to protocol problems of genuine utility. He is also interested in social and ethical issues connected to technology.