# An Obsession with Definitions

Phillip Rogaway(✉)

Department of Computer Science, University of California, Davis, USA
rogaway@cs.ucdavis.edu

**Abstract.** Many people seem to think that cryptography is all about creating and analyzing cryptographic schemes. This view ignores the centrality of *definitions* in shaping the character of the field. More than schemes or their analysis, it is definitions that most occupy my thoughts. In this paper, written to accompany an invited talk at Latincrypt 2017, I try to explain my own fascination with definitions. I outline a few of the definitions I've recently worked on—garbling schemes, online AE, and onion encryption—and provide some general advice and comments about the definitional enterprise.

**Keywords:** Cryptographic definitions · Garbling schemes · Onion encryption · Online authenticated-encryption · Provable security

## 1 Introduction

Cryptography is about more than creating and analyzing cryptographic schemes. While these two activities are central cryptographic tasks, cryptography is also, and importantly, about definitions.

Cryptographic definitions emerged within the provable-security framework, which largely begins with Goldwasser and Micali [18]. The basic steps are to *define* a cryptographic problem, to devise a *protocol* for it, and to *prove* that the protocol satisfies its definition, assuming that some other cryptographic scheme satisfies *its* definition. This type of proof is known as a *reduction*.

It seems little discussed that cryptographic definitions can be significant beyond the provable security framework. (a) Definitions can have a profound role in what we *see* in our field—what is rendered visible—and how we approach working on problems [23]. (b) Definitions can enable clear communication and clear thinking. When you encounter confusion in cryptography—which is often—the root cause is often a lack of agreement as to what you're trying to accomplish, and what the words even mean. (c) Definitions can be useful in breaking schemes. I remember breaking the NSA's Dual Counter Mode encryption scheme [10] in minutes, as I read the spec. The NSA claimed this authenticated encryption (AE) scheme to be the product of a 1.5 year effort. It's not that I'm a skilled cryptanalyst—I am not. What I had that the NSA authors obviously didn't was an understanding of a definition for AE. (d) Definitions can give rise to schemes with improved efficiency. When I started off, I anticipated that there would be a


T. Lange and O. Dunkelman (Eds.): LATINCRYPT 2017, LNCS 11368, pp. 3–20, 2019.
https://doi.org/10.1007/978-3-030-25283-0_1

cost, in running time, to doing things with definitions and proofs. And sometimes there is. But, just as often, the exact opposite happens. By having definitions and proofs you are sometimes able to develop mechanisms that let you "cut to the bone," but no deeper, thereby improving efficiency.(e) Finally, there are definitional models that fall outside of the provable security paradigm, such as work done purely in the random-oracle model, the random-permutation model, or the ideal-cipher model; or things in Dolev-Yao style models.Such work absolutely *is* cryptography, begins with definitions, and shouldn't be denigrated because it doesn't fall within the reduction-based tradition.

My plan for this paper is to provide examples of definitions in three different domains. I've chosen definitions that are relatively recent, and that have something to do with encryption. I'll discuss *garbling schemes* (or *garbled circuits*), *online AE*, and *onion AE* (or simply *onion encryption*). The hope is that by giving multiple examples I'll manage to communicate something about the character of definitional work that I wouldn't manage to communicate if I spoke more abstractly, or if focused on a single example. My conclusions, supported by the three examples, are in Sect. 5.

I first wrote about the value of cryptographic definitions more than a decade ago [22]. The current paper is from a different perspective and uses different examples. The examples are from papers with Mihir Bellare, Viet Tung Hoang, Reza Reyhanitabar, Damian Vizár, and Yusi Zhang [4,19,25,26].

## 2 Garbling Schemes

In a conventional boolean circuit, a *label*, zero or one, is associated to each wire. These represent truth values, the 0-label corresponding to false and the 1-label corresponding to true. The labels propagate up the circuit, moving from the input wires toward a designated output wire. If you possess the labels for the inputs to some gate then you possess the label for its output, which you compute according to the functionality of the gate. An or-gate with input labels of 0 and 1 gives you an output label of 1; an and-gate with input labels of 0 and 1 gives you an output label of 0; and so on.

A *garbled circuit* is similar, but instead of propagating labels with known semantics you propagate *tokens* with unknown semantics. There are two tokens associated to each wire. They are strings, maybe 128 bits each. You can think of them as random 128-bit strings. Possession of one token doesn't imply what the other one is. To be sure, there is a semantics, a truth values, associated to each token; if a wire has tokens of $A$ and $B$, either $A$ represents true and $B$ represents false, or else it's the other way around. But you can't just look at a token and know what its semantics is.

If you happen to possess an $A$ token (of $A$ and $B$) for the left-hand wire of an and gate, and you have the $C$ token (of $C$ and $D$) for the right-hand, and if the output wire has tokens of $E$ and $F$, then *if* $A$ has hidden semantics of false and $C$ of true, then you need to be able to compute the output token, whether $E$ or $F$, that has hidden semantics of false. In general, each gate, which you can

call a *garbled gate*, is a little algorithm, a recipe, that tells you how to propagate tokens from the input wires to the output wire, doing this in a way that respects the hidden semantics. You evaluate the garbled circuit in this manner until you acquire a token for the final output, and usually the semantics associated to that particular token is made manifest from the token itself. This is how evaluating a garbled circuit works.

Garbled circuits were first described in oral presentations given by Andy Yao in the early or mid 1980s, and it become customary to cite a certain 1986 paper of Yao's for the garbled-circuit idea [30]. But if you read the paper, you won't find there any hint of garbled circuits. I'm actually not sure if the conventional citation is even the paper corresponding to Yao's talks; a 1982 paper of Yao's looks to be more relevant [31], although there's nothing described there like garbled circuits, either. Perhaps the informal and oral culture surrounding this area is part of what launched garbled circuits on a trajectory in which definitions weren't seen as essential. This became characteristic of multiparty computation (MPC) more broadly, despite it being embraced by the theoretical computer science, STOC/FOCS, community.

You might guess that in the intervening 25 years, *someone* would have gone back and defined just what a garbled circuit was intended to do. Strangely, this didn't happen. I think the reason for this is that an entire community had come to view garbled circuits as a *technique*. Being seen as a *method* for solving other problems, it wasn't conceptualized as something in need of a definition.

Originally, garbled circuits had been seen as a tool to solve two-party secure-function evaluation. One party, call it $B$, presents to the other party, call it $A$, a garbled circuit specialized to his own (that is, $B$'s) input. The sending party $B$ also helps the receiving party $A$ to acquire exactly one of the two tokens for each input wire held by $A$—in particular, to acquire the token with the *correct* semantics for that wire, as per $A$'s input. Party $A$ then evaluates the garbled circuit on its own, pushing tokens up the circuit until it has the garbled output, which, as I said before, was supposed to be interpretable as an actual output.

So my coauthors and I wanted to identify the problem that garbled circuits are implicitly intended to solve [4]. I think that one of the key realizations was that garbled circuits really have nothing to do with circuits. In fact, there were already examples in the literature in which people were garbling other things—arithmetic circuits, for example, or branching programs, or DFAs. More recently, people have been wanting to garble Random Access Machine (RAM) computations, or Turing Machine computations. The part of garbling that one really needed to focus on wasn't the garbled *thing*, but the garbling *scheme* that produces it—the process that takes in a function and turns it into a garbled version of that function. This should be done in a way that's representation-independent. That was our first goal for a definition in this domain. A second goal was to make sure that we would encompass as many applications as possible. There are already over a thousand papers that used garbling and we weren't going to be able to have an abstraction that would work for all of them. But we wanted something that would work for most.

So here is an overview for our definition of a garbling scheme. First, and quite importantly, we have to lay down the *syntax* for what a garbling scheme does. When I speak here of syntax, I mean the types of algorithms one must specify for a scheme—the signature (the domain and range) for each. By way of example, if you are defining a blockcipher, the syntax is that of a map $E$: $\mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ such that $E(K,\cdot)$ is always a permutation. It seems really useful to separate the syntax of the object you're defining from the measure of security for it.Yet it's common for people to try to define cryptographic goals without attending much to the desired syntax. This is a mistake.

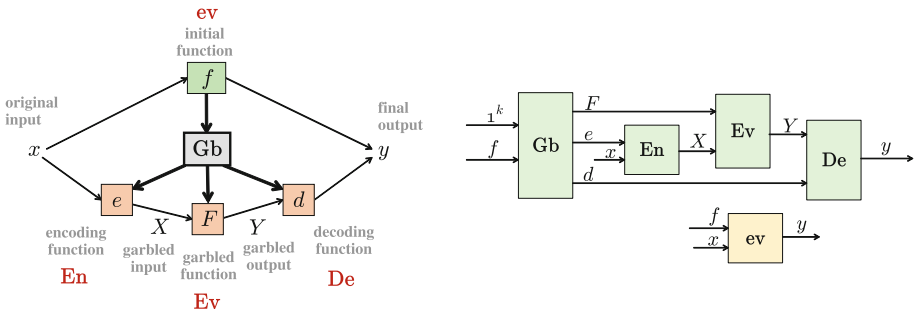


**Fig. 1. Garbling schemes. Left:** Informally, a function $f$ is probabilistically factored into a triple of functions $d \circ F \circ e$. In actuality, all of these are strings. They represent functions using the maps ev, En, Ev, and De. **Right:** Formally, a garbling scheme consists of functions (Gb, En, De, Ev, ev) satisfying the indicated relationship. The function Gb is probabilistic; the remaining functions are deterministic.

So here's the basic idea for a garbling scheme's syntax. See the left-hand side of Fig. 1. You're given some function $f$—think of it as the *initial* function—and *garbling* it is a way of "factoring" $f$, probabilistically, into three pieces: $f = d \circ F \circ e$ (with composition written right-to-left). We're going to feed $f$ (maybe a circuit you want to garble) into a *garbling scheme* that will produce those three pieces: an *encoding function* $e$, which will take in the *initial inputs* and produce the *garbled inputs*; the *garbled function* $F$ itself; and the *decoding function* $d$, which will take in the *garbled output* and produce the *final output*. We intend that if $f$ maps $x$ to $y$, then you will get the same result by feeding $x$ into $e$ to get the garbled input $X$, then feeding $X$ into the garbled function $F$ to get the garbled output $Y$, then feeding $Y$ into decoding function $d$ to get $y$.

A problem with what I've just described is that I am treating $f$, $e$, $F$, and $d$ as both functions and as strings that are either fed into algorithms or spat out by them. That doesn't work in this setting, because the whole point of garbling is to deal with matters of how things are represented. We therefore need to be quite explicit about how we interpret strings as functions. In many contexts in cryptography we don't need to bother with that; we treat strings as describing functions with very little fuss. But here we're going to regard $f$, $F$, $e$, and $d$ as

strings that describe functions under the auspices of maps named ev, En, Ev, and De, respectively. See the right-hand side of Fig. 1, where a garbling scheme is shown to be a 5-tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. The main algorithm, the "business end" of things, is the probabilistic garbling algorithm Gb. It maps a string representing a function to a string representing a garbled function. Then there are four more functions. These encode the initial input, decode the garbled output, evaluate the string representing the garbled function, and evaluate the string representing the initial function. We intend first a *correctness condition*, as illustrated on the left side of Fig. 1. You always get the same thing when you go through the route on the top and the route on the bottom. There's also a *nondegeneracy* condition that makes sure that the main work is done in the garbling function itself; we don't want you to factor $f$ in such a way that the real work is happening in the encoding or decoding step.

What I've described so far is the syntax of a garbling scheme; I haven't described security. In our paper [4], we define notions we call *privacy*, *authenticity*, and *obliviousness*. I'll only describe the first. The basic intuition for privacy is that learning $(F, X, d)$—the garbled function, the garbled input, and the decoding function—shouldn't reveal anything except the final output $y$. But the basic intuition isn't quite right, because real garbling schemes always leak something beyond the output $y$. A garbled circuit typically leaks the topology of the underlying circuit—the pattern of how gates are connected up—if not the circuit itself. You could hide these things, but you would still leak the circuit's size. In order to capture what is understood to be leaked, we provide a *side-information function*, denoted $\Phi$. Given an initial function $f$, it indicates the information $\Phi(f)$ that we expect to leak by revealing $F$. For circuit garbling, side-information functions $\Phi_{\text{circ}}(f)$, $\Phi_{\text{topo}}(f)$, and $\Phi_{\text{size}}(f)$ return all of $f$, the topology of $f$, and the size of $f$, respectively. We always assume that we leak at least the *signature* of the function $f$, meaning the values $n$ and $m$ where $f\colon \{0,1\}^n \to \{0,1\}^m$.

Now ready to define privacy for a garbling scheme, we imagine that an adversary is presented one of two types of oracles—a left-handed oracle or a right-handed oracle. Which type of oracle the adversary gets is determined by a coin flip $b \twoheadleftarrow \{0,1\}$: if $b = 0$ the adversary gets a left-handed oracle; if $b = 1$ it gets a right-handed oracle. Our adversary presents to its oracle a pair of pairs $(f_0, x_0)$, $(f_1, x_1)$. If the oracle is left-handed then it properly garbles the left-hand pair: it computes $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, f_0)$ and then sets $X \leftarrow \mathsf{En}(e, x_0)$. If the oracle is right-handed then it garbles the right-hand pair: it computes $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, f_1)$ and then sets $X \leftarrow \mathsf{En}(e, x_1)$. Either way, the oracle now returns $(F, X, d)$: the garbled function, the garbled input, and the decoding function. The adversary's job is to guess the bit $b$. We do have to ensure that $f_0(x_0) = f_1(x_1)$; otherwise, it would be easy for the adversary to compute the value $b$. We also have to ensure that the side-information doesn't allow a trivial distinguishing attack. Formalizing these requirements, regardless of $b$, the oracle begins by testing if $\Phi(f_0) \neq \Phi(f_1)$ or $\mathsf{ev}(f_0, x_0) \neq \mathsf{ev}(f_1, x_1)$. If so, the oracle just returns a distinguished symbol $\perp$. The adversary's advantage $\mathbf{Adv}^{\text{prv.ind}}_{\mathcal{G},\Phi}(\mathcal{A}, k) = 2\Pr[b' = b] - 1$ is the probability that $\mathcal{A}$ correctly identifies $b$, renormalized, as usual, to the interval $[-1, 1]$.

There is also a simulation-based notion for garbling privacy. Here the adversary presents to its oracle a single pair $(f, x)$. The oracle will behave in one of two ways, based on a coin flip $b \twoheadleftarrow \{0,1\}$. If it's a "real" oracle, $b = 1$, then it garbles the function $f$ and its input $x$, computing $(F, e, d) \twoheadleftarrow \mathsf{Gb}(1^k, f)$ and $X \leftarrow \mathsf{En}(e, x)$. It returns $(F, X, d)$. When the oracle is a "fake" oracle, $b = 0$, a simulator must provide $(F, X, d)$ without benefit of seeing $f$ and $x$. What it is given instead is $\Phi(f)$ and $f(x)$. This definition of privacy also seems like a relatively direct way to capture the idea that the garbled input, the garbled function, and the decoding function don't leak anything that shouldn't be leaked. Adversarial advantage is defined as before, $\mathbf{Adv}^{\mathrm{prv.sim}}_{\mathcal{G},\Phi,S}(\mathcal{A}, k) = 2\Pr[b' = b] - 1$.

How do these two notions relate? They're quite close. The simulation definition implies the indistinguishability version, and the indistinguishability version implies the simulatability version if you add in some modest side-condition.

Where does this go from here? By having definitions for what a garbling scheme is supposed to do we are able to achieve much improved efficiency [2]: we can do garbled circuit evaluation, in the ideal-permutation model, where the work associated to evaluating a gate is, say, a single AES computation employing a fixed key. In practice, something like 25 clock cycles, or 7.5 ns, per gate. Much of this efficiency improvement, however, isn't due to any cryptographic advance, but to a different definitional aspect: formalizing circuits in a particularly clean way, then implementing directly to that formalization. In another direction, we look at *dynamic* adversaries (they corrupt parties as-they-go) [3]. We find that garbled circuits, as conventionally realized, don't work for dynamic adversaries. When papers implicitly assumed that conventional circuit garbling works even in the case of dynamic adversaries, they made claims that were not correct.

## 3 Online AE

I will next describe online authenticated-encryption (online AE), sketching joint work with Hoang, Reyhanitabar, and Vizár from 2015 [19].

I hope that readers have seen the "basic" notion of AE, what I'll call nonce-based AE (NAE). Under NAE, encryption is a deterministic way to make a key $K$, associated data $A$, and a message $M$ to a ciphertext $C$; formally, an NAE scheme is a function $\mathcal{E} \colon \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ where $\mathcal{E}(K, N, A, \cdot)$ is an injection, where $x \in \mathcal{M}$ implies $\{0,1\}^{|x|} \subseteq \mathcal{M}$, and where $|\mathcal{E}(K, A, M)| = |M| + \tau$ for some constant $\tau$. We can then define the decryption function from the encryption function, letting $\mathcal{D}(K, N, A, C) = M$ if there is an $M$ for which $\mathcal{E}(K, N, A, M) = C$, and letting $\mathcal{D}(K, N, A, C) = \perp$ otherwise. Since we have defined the behavior of $\mathcal{D}$ from $\mathcal{E}$, we can omit the usual correctness condition, which would mandate that they appropriately compose.

For NAE security, we ask for the following indistinguishability condition [24]. In the "real" game, an adversary is given an oracle that encrypts according to $\mathcal{E}_K$: for a $K$ chosen uniformly from $\mathcal{K}$ at the beginning of the game, it responds to a query $(N, A, M)$ with $\mathcal{E}_K(N, A, M)$. It also has a second oracle that decrypts according to $\mathcal{D}_K$: given $(N, A, C)$, it returns the plaintext $M = \mathcal{D}_K(N, A, C)$.

We ask the adversary to distinguish this pair of oracles from an "ideal" pair of oracles—the first of which spits out $|M|+\tau$ random bits and the second of which always answers $\perp$, an indication of invalidity.

When you make a definition like this, you need to add in provisos to outlaw trivial wins. The adversary could always encrypt some $(N, A, M)$ to get $C$, then decrypt $(N, A, C)$ to get back either $M$ or $\perp$, thereby identifying the operative game. Similarly it could repeat a nonce value $N$ in an encryption query, violating the intent of a nonce. So we must either forbid these activities, or give the adversary no credit if it engages in them. The advantage is defined as $\mathbf{Adv}^{\mathrm{nae}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathcal{E}_K, \mathcal{D}_k} \to 1] - \Pr[A^{\$, \perp} \to 1]$. This NAE notion has been quite influential. For example, the CAESAR competition going on right now [7] has resulted in dozens of submissions designed to meet it.

Now one of the often-heard complaints about NAE is that a conforming scheme can fail completely if nonces are reused. And we know from experience that nonces *do* get reused with alarming regularity. A recent example is the KRACK attack on WPA2 [29]. Nonce reuse is usually a result of error, but there are also settings in which it is difficult or impossible to ensure that nonces don't get reused. So a nice strengthening of the NAE notion is the notion of *misuse-resistant* AE, or MRAE [24]. It's not a great name, as the misuse we are considering here is only one form of misuse, namely, nonce reuse.

The definition of MRAE looks just like the definition of NAE except that we allow the adversary to repeat $N$-values on encryption queries, as long as it doesn't repeat an entire $(N, A, M)$ triple. That's it. Yet the notion becomes quite different to achieve. One of the key differences between NAE and MRAE is that, for the latter, you've got to read the entire plaintext before you can output even the first bit of ciphertext: online encryption is impossible. This is easy to see because if the first bit of ciphertext didn't depend on the last bit of plaintext then you'd have an easy way to win the MRAE game.

There are situations where the can't-be-online restriction is a problem. You might have a long input an no ability to buffer it all. You might need to act on the prefix of a message before hearing the suffix. There has therefore been a perceived need to achieve something *like* MRAE, but where online encryption would be possible. When I say that online encryption is possible, I mean that you should be able to read in a plaintext left-to-right, and, using a constant amount of memory, spit out bits until you extrude the entire ciphertext.

In order to answer this need, Fleischmann, Forler, and Lucks (FFL) proposed a security notion for online authenticated-encryption [15] that I'll call OAE1. It was said to imply security against nonce reuse, yet to be achievable by online schemes. The notion quickly caught on. Among the round-1 CAESAR submissions (2014), fully a dozen claimed OAE1 security. Additional schemes were said to achieve something closely related to OAE1 security. This is an extraordinary degree of influence for a 2012 definition.

But there are problems with OAE1. To define the notion, I must go back to an earlier definition from Bellare, Boldyreva, Knudsen, and Namprempre for an *online cipher* [1]. They start by fixing a block length $n$ and assuming that

we're going to be enciphering strings that are a multiple of $n$ bits. They imagine an ideal object in which the $i$-th $n$-bit block of plaintext gets enciphered to the $i$-th $n$-bit block of ciphertext. This block may depend only on blocks 1 to $i$. A good online cipher must approximate this ideal object. More formally, let $\mathsf{B}_n = \{0,1\}^n$ and define a *multiple-of-n cipher* as a map $\mathcal{E}\colon \mathcal{K} \times \mathsf{B}_n^* \to \mathsf{B}_n^*$ where $\mathcal{E}(K,\cdot)$ is a length-preserving injection for each $K$. Let $\mathrm{OPerm}[n]$ be all multiple-of-$n$ ciphers $\pi$ where the $i$-th block of $\pi(X)$ depends only on the first $i$ blocks of $X$. A good online cipher is a multiple-of-$n$ cipher $\mathcal{E}$ where $\mathcal{E}(K,\cdot)$ is indistinguishable from a random permutation $\pi \twoheadleftarrow \mathrm{OPerm}[n]$.

FFL's definition for OAE1 modifies the online-cipher notion in a simple way. FFL again assume that $n$ is fixed and that the plaintext is a multiple of $n$ bits. They assume that the output is going to look like a ciphertext piece that is exactly as long as the original plaintext, followed by a $\tau$-bit authentication tag $T$. The ciphertext piece must be given by an online cipher, although tweaked by a header that encodes the nonce $N$ and associated data $A$. The tag $T$ should be pseudorandom: it should look like $\tau$ random bits. This is the privacy notion. There is also a standard notion of authenticity that goes with it, capturing unforgeability under an adaptive chosen-message attack.

Unfortunately, the OAE1 definition doesn't make a whole lot of sense. Its problems include:

1. *Admits unexpected attacks.* OAE1 doesn't guarantee the anticipated degree of privacy in the face of nonce reuse. With an OAE1-secure scheme, if you're given an oracle that takes in a message-prefix, appends a secret suffix, and then encrypts the whole thing, then you can play with that oracle to recover that secret suffix, in its entirety, with a small number of queries. Attacks like this can be damaging, as seen in the BEAST attack on SSL/TLS [14]. Still, this is not the worst problem for OAE1. Maybe it's just incorrect intuition about what online AE can be expected to achieve; maybe any online-AE scheme must fall to this sort of attack.
2. *The blocksize should be user-selectable, not a scheme-dependent constant.* Why were we trying to make an online-AE scheme in the first place? It is due to some memory or latency restriction. Maybe the encryption is realized by an ASIC or FPGA, and we can only give over so much memory for the task. Maybe a video file is being streamed to a user, and there is only so long the user should wait. Whatever the restriction is, it has nothing to do with the blocksize of some implementing blockcipher. That value, which is likely to be quite small (like or 64 or 128 bits), has nothing to do with the extent to which a scheme can buffer things or wait. Since we can't realistically know what the user's actual constraint is, it is best if this is left unspecified—a user-selectable value, not a scheme-dependent constant.
3. *Decryption also needs to be online.* It's very strange to demand that encryption be online but make no analogous requirement for decryption. In fact, it's hard to even think of a context in which it would be OK for decryption to require buffering the whole message, and yet there was a constraint that this not be done at encryption time. The OAE1 notion effectively demands

that decryption *not* be online, as the authenticity formalization effectively demands that no information be released until the ciphertext, in its entirety, is checked for validity.

4. *Security needs to be defined for strings of all lengths.* The OAE1 definition inherits from the definition of an online cipher the peculiar characteristic that the length of the input *must* be a multiple of $n$. When you actually describe a general-purpose scheme you're going to want to ensure that it works on all bit strings, or at least all byte strings. So we need a security definition that is applicable to all bit strings or all byte strings. And saying that you achieve a rich domain through padding begs the question: what exactly is it that you are achieving through the use of padding?
5. *The reference object is not ideal.* Finally, the reference object that OAE1 measures a scheme against is not ideal: one can do better than something that looks like an ideal online cipher with a small $n$ followed by a bunch of random-looking bits. We should not be taking such an object as our yardstick.

In order to address these concerns we need to change not only the OAE1 security notion, but to completely revise the basic syntax.

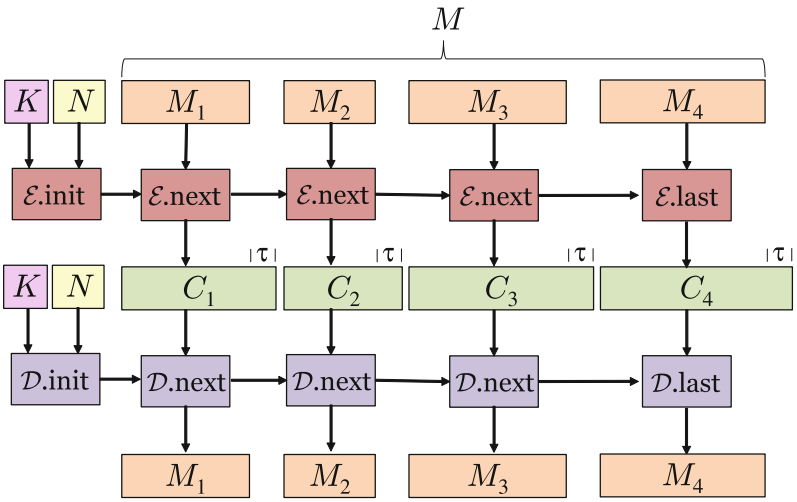


**Fig. 2. Operation of an OAE2-secure AE scheme.** The plaintext is broken into segments $M = M_1 M_2 \cdots M_m$ of arbitrary and possibly varying lengths. Each $M_i$ is encrypted to a ciphertext segment $C_i$ of length $|M_i| + \tau$. The key $K$ and nonce $N$ initialize the process. Decryption reverses the process. If an authenticity error should arise, it continues for all subsequent blocks. The figure omits associated data.

Refer to Fig. 2. Messages to be encrypted or decrypted will be *segmented*. We think of the encryption scheme's user as selecting the segmentation. To be fully general, we allow the segments to have varying lengths (although we

expect that, typically, all segments but the last will have the same length). Segmenting the plaintext amounts to supporting an API that says: here's the first piece of message I'd like to encrypt; and here's the next piece; and so on, until one presents the final message chunk. We would normally expect that each ciphertext segment $C_i$ has a length $|M_i| + \tau$ that is a fixed amount greater than the corresponding plaintext segment. We expect that each plaintext $M_i$ will fit in memory, but the concatenation of all plaintext segments might not. After processing a plaintext segment, the encryption algorithm can remember what it wants by updating its constant-size internal state. The initial state is determined by the key $K$ and the nonce $N$.

Decrypting the segmented ciphertext $(C_1, C_2, \ldots, C_m)$ that arose from encrypting $(M_1, M_2, \ldots, M_m)$ should result in the original segmentation, while changing the segmentation in any way should result in an authentication failure from that point on. If a ciphertext segment gets corrupted, this should result in an authentication failureon trying to decrypt that segment, and decryption of all future segments should also fail.

Here now is the syntax for what we want. We say that a *segmented-AE scheme* is a triple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where $\mathcal{K}$ is a distribution on strings (or the algorithm that induces it), and $\mathcal{E} = (\mathcal{E}.\text{init}, \mathcal{E}.\text{next}, \mathcal{E}.\text{last})$ and $\mathcal{D} = (\mathcal{D}.\text{init}, \mathcal{D}.\text{next}, \mathcal{D}.\text{last})$ are triples of deterministic algorithms where $\mathcal{E}.\text{init}\colon\ \mathcal{K} \times \mathcal{N} \to \mathcal{S}$ and $\mathcal{E}.\text{next}\colon\ \mathcal{S} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C} \times \mathcal{S}$ and $\mathcal{E}.\text{last}\colon\ \mathcal{S} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ and $\mathcal{D}.\text{init}\colon\ \mathcal{K} \times \mathcal{N} \to \mathcal{S}$ and $\mathcal{D}.\text{next}\colon\ \mathcal{S} \times \mathcal{A} \times \mathcal{C} \to (\mathcal{M} \times \mathcal{S}) \cup \{\perp\}$ and $\mathcal{D}.\text{last}\colon\ \mathcal{S} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$. For simplicity, assume $\mathcal{A} = \mathcal{M} = \mathcal{C} = \{0,1\}^*$ and $\mathcal{N} \subseteq \{0,1\}^*$. For generality, we have allowed a new segment $A_i$ of associated data to be presented alongside each new segment $M_i$ of plaintext.

How do we define security? The notion we call OAE2 can again be described with a simple pair of games. The "real" game looks as we just described encryption: the adversary can segment messages however it likes, it can select arbitrary nonces, and for each such nonce $N$ and $(M_1, \ldots, M_m)$ it gets back the corresponding segmented ciphertext $(C_1, \ldots C_m)$. It can repeat $N$-values in any manner it likes. For the "ideal" game, which the adversary must distinguish from the "real" one, the $\mathcal{E}$.next functionality is replaced by a random injective function $f$, tweaked as we will describe, mapping $\ell$-bit strings to $(\ell + \tau)$-bit strings for all $\ell$. Now as for the tweaks:the function $f$ used to map the first segment is tweaked by $N$, so $C_1 = f_N(M_1)$; the function $f$ used to map the second segment is tweaked by $N$ and $M_1$, so $C_2 = f_{N,M_1}(M_2)$; the function $f$ used to map the third segment is tweaked by $N$, $M_1$, and $M_2$, so $C_3 = f_{N,M_1,M_2}(M_3)$; and so on. The last block is different. It is important to distinguish the last block from prior ones, as the *end* of a message is a distinguished status. For the last block, then, for the ideal functionality corresponding to what is processed in the "real" scheme using $\mathcal{E}$.last, we set $C_m = f_{N,M_1,M_2,\ldots,M_{m-1},*}(M_m)$, the $*$ denoting another variant tweak. Throughout, distinct tweaks correspond to independent random injections. For simplicity, our description has omitted the associated data.

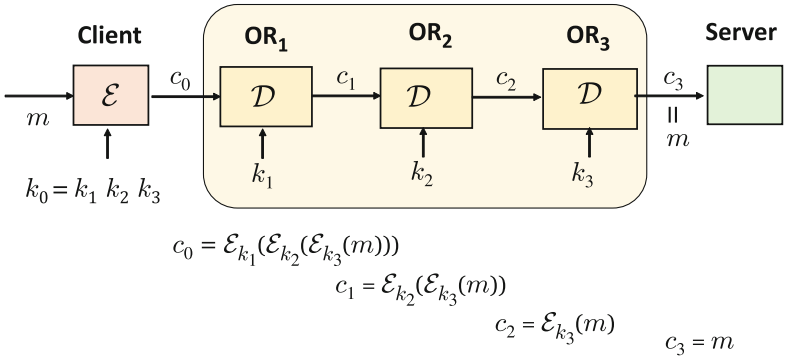


**Fig. 3. The nested-encryption approach for onion routing.** The client shares a key $k_i$ with each onion router $\mathrm{OR}_i$. To send a message $m$ to a server, the client iteratively encrypts it. Each OR will "peel off" (decrypt) a single layer of encryption. The nested-encryption idea, for the public-key setting, is from Chaum [11].

The development of OAE2 would now continue by exploring definitional variants, then investigating schemes that meet the notions. I don't think it particularly desirable to create schemes for a task like this from scratch; better to start with an NAE or MRAE scheme.

In describing where OAE2 came from, I told it as a story of wanting to fix the OAE1 definition from FFL. But I could have arrived at the same definition from other directions. The kind of online AE goal we are after has been known for a long time. Prior work by Tsang, Solomakhin, Smith [28] and by Bertoni, Daemen, Peeters, Van Assche [8] gives related notions and schemes. Netflix has posted a protocol description for an object that does virtually the same thing as we do [20]. In general, for questions of practical importance in cryptography, practitioners will often have noticed the problem and posed solutions long before it gets on the radar of any theory-minded cryptographer.

## 4 Onion Encryption

The last definitional problem I'd like to describe is that of *onion encryption*, or *onion-AE*. This is recent work with Yusi Zhang [25,26].

The task we address is another type of AE. See Fig. 3. A *client* (or *user*) wants to send a message to a *server* (or *destination*), transmitting it over an *onion-routing network*, which is a collection of cooperating servers (onion routers). In the usual solution, the client will iteratively encrypt the message $m$ it wants to send, once for each intermediate OR (onion router). So, in the figure, the plaintext $m$ is turned into a ciphertext $c = E_{k_1}(E_{k_2}(E_{k_3}(m)))$. The ORs will each decrypt the ciphertext they receive, so that, at the end, the plaintext $m$ will be recovered. The idea goes back to the early development of Tor [13,16,17,27] and, reaching back further still, to the idea of mixnets, from David Chaum [11].

As with garbling schemes, nested encryption has been understood as technique, not a solution to some clearly articulated problem. The question Zhang and I asked is: just what goal is it that nested encryption aims to solve?

Of course it is easy to answer such questions in vague English. The client is trying to get some message over to the server in such a way that individual ORs won't know the association between the sender and the receiver. They shouldn't even know who is the client (except for the initial OR) and who is the destination (except for the final OR). We want to make it hard for an adversary that has less than a total view of the network to know who is communicating with whom. We hope that bogus ciphertexts inserted into the network will not emerge from the exit node. One can go on in such a vein, and it is not useless, but there is a large gap between this level of discourse and a cryptographic definition.

We begin, as always, by formalizing the syntax for the object we are after. We say that an *onion-encryption scheme* is a triple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ as follows. (a) The key-generation algorithm $\mathcal{K}$ is a probabilistic algorithm that maps a number $n$ (the number of ORs) to a vector of $n+1$ strings: $(k_0, k_1, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$. The first key in the vector is intended for the client; the next $n$ keys are intended specific the ORs, key $k_i$ for router $\mathrm{OR}_i$. (b) The encryption algorithm $\mathcal{E}$ is a deterministic algorithm that maps a key $k_0$, a message $m$, and the user's state $u$ to a ciphertext $c_0$ and an updated state $u'$. It specifies what the client must do to encrypt a message. It is important that $\mathcal{E}$ is stateful: each time the client encrypts a message, its internal state gets updated. The initial state is the empty string. (c) The decryption algorithm $\mathcal{D}$ is a deterministic algorithm that maps a key $k_i$, a ciphertext $c$, and an OR's current state $s_i$ to something that is either a plaintext $m'$, a ciphertext $c'$, or the symbol $\diamondsuit$, an indication of failure. Decryption also produces an updated state $s_i'$. The decryption algorithm specifies what an OR must do on receipt of a ciphertext $c$. Only ciphertext outputs (or $\diamondsuit$) can be produced using keys from $k_1, \ldots, k_{n-1}$, and only plaintext outputs (or $\diamondsuit$) can be produced using the key $k_n$. Throughout, plaintexts and ciphertexts are recognizably distinct, the former coming from a message space of $\{0,1\}^{l_1}$ and the latter coming from a ciphertext space of $\{0,1\}^{l_2}$ with $l_2 > l_1$.

After defining a scheme's syntax one typically specifies a *correctness condition*. It captures the fact that, in the absence of an adversary, everything works as you expect. Formally, we assert that for every number $n$ and every vector of strings $m \in \{0,1\}^{**}$, if $k \twoheadleftarrow \mathcal{K}(n)$ is a vector of keys then the following predicate $\mathrm{Correct}_{\Pi}(k, m)$ is true:

**procedure** $\mathrm{Correct}_{\Pi}(k, m)$
$(k_0, k_1, \ldots, k_n) \leftarrow k$; $(m_1, \ldots, m_\ell) \leftarrow m$; $u, s_1, \ldots, s_n \leftarrow \varepsilon$
**for** $i \leftarrow 1$ **to** $\ell$ **do**
  $(c_0^i, u) \leftarrow \mathcal{E}(k_0, m_i, u)$
  **for** $j \leftarrow 1$ **to** $n$ **do** $(c_j^i, s_j) \leftarrow \mathcal{D}(k_j, c_{j-1}^i, s_j)$
**return** $\bigwedge_{1 \le i \le \ell} (m_i = c_n^i)$

The goal we aim to formalize looks like stateful AE [6,9], but with the decryption done by the OR network itself. Privacy will be formalized in the tradition of indistinguishability from random bits. This will make for an easy way to get the desired anonymity property. The authenticity property we aim for is authenticity checked at time of exit. One can, alternatively, check authenticity earlier and repeatedly, at each OR. This notion might be more desirable, but it's not how Tor's relay protocol does things [12,13], so it's not the approach we take, either.

One can write a pair of games that directly corresponds to what we've sketched. They would look something like this. The "real" game would run an Initialize routine that, given the number $n$ of ORs, would initialize keys $(k_0, k_1, \ldots, k_n) \twoheadleftarrow \mathcal{K}(n)$. The game would support oracle calls of $\mathrm{Enc}(m)$ and $\mathrm{Dec}(i, c)$. A call to $\mathrm{Enc}(m)$ would compute $(c, u) \leftarrow \mathcal{E}(k_0, m, u)$ then return $c$. A call to $\mathrm{Dec}(i, c)$ would compute $(m, s_i) \leftarrow \mathcal{D}(k_i, c, s_i)$ then return $m$. In contrast, in the "ideal" game a call to $\mathrm{Enc}(m)$ would return the appropriate number of random bits, while a call to $\mathrm{Dec}(i, c)$ would return the appropriate number of random bits if $i < n$, and an indication of invalidity $\diamondsuit$ if $i = n$.

So far, nothing I've said is unexpected or complex. But, left as is, it is also wrong. If an adversary were to first encrypt a message $m$ for a sequence of $n = 3$ ORs, getting back a ciphertext $c_0 \leftarrow \mathrm{Enc}(m)$, and if it were to next perform the sequence of decryption calls $c_1 \leftarrow \mathrm{Dec}(1, c_0)$, $c_2 \leftarrow \mathrm{Dec}(2, c_1)$, $c_3 \leftarrow \mathrm{Dec}(3, c_2)$, then the adversary would have a trivial way to win: by returning 1 if and only if $c_3 = m$. Intuitively, this sort of win shouldn't count, for all the adversary did was to use the OR network to decrypt a plaintext it knew the decryption of. But how, precisely, do we forbid the adversary from scoring this sort of win? What does *this sort of win* actually mean? Formalizing it is rather complex. The code gets complicated enough that it's hard to verify if it's right—and none to easy to use, either. Perhaps this is emblematic of a wider problem we face in dealing with complex cryptographic definitions. We're pretty good with "simple" definitions, but as you move up to harder tasks, formalizing things can get unconvincing and obscure.

Might there be a better way? I'd like to describe the approach that Zhang and I have been working on [25,26]. It is offered as a general framework for doing indistinguishability definitions in settings where an adversary must distinguish a "real" and an "ideal" world. This isn't much of a constraint, as success-style games, where the adversary wants to induce some event to happen, can be rewritten in the real/ideal format. We will think of the real and ideal worlds as being described by pieces of pseudocode.

In our onion-encryption example, and in other examples of this paper, the simple indistinguishability-based definition you would *like* to write down is not achievable; there exists some simple adversarial attack which *would* distinguish between the two worlds. Usually we use our intuition to inform us as to what restrictions are necessary so as to not credit these trivial attacks. You can either say "the adversary's not allowed to do *this*"—you pose it as a restriction on the class of acceptable adversaries—or, alternatively, you let the adversary do what it wants, but, at the end of the game, you have a "finalization procedure" look

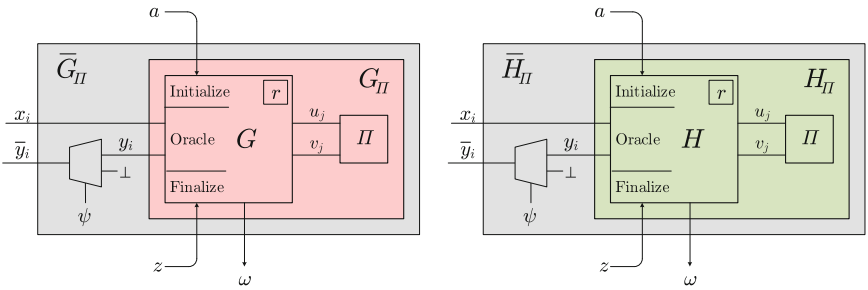


**Fig. 4. Oracle silencing.** Game $G$ captures the *real* environment; game $H$, the *ideal* one. Both depend on some scheme $\Pi$. An adversary (not shown) makes a sequence of *oracle calls*. In the *silenced* games $(\overline{G}, \overline{H}) = \text{Silence}_C(G, H)$, responses are suppressed according to predicate $\Psi = \mathit{Fixed}_{C,G}(x_1, y_1, \ldots, x_k, y_k, x)$, which is true if there exists a unique $y$ such that $\mathit{Valid}_{C,G}(x_1, y_1, \ldots, x_k, y_k, x, y)$. The *Valid* predicate asserts that there is some $\Pi \in C$ that can give rise to the indicated partial transcript.

back at what transpired and *if* the adversary misbehaved, you penalize it—you deny it credit for the win. Bellare, Hofheinz, and Kiltz call these two approaches *exclusion-style* definitions and *penalty-style* definitions [5]. They carefully investigate two versions of each for the problem of IND-CCA secure public-key encryption (PKE).

I'd like to suggest a third possibility. We put a little bit of smarts in the oracle itself, so that the oracle recognizes that it's about to provide the caller an answer that would give away the show. When this happens, the oracle shuts up. It returns a distinguished value $\perp$. This happens in both the real and the ideal game. We call this style of game modification *oracle silencing*. In our early work, the oracle would shut up just for the one query. Now we've been doing things so that the oracle, once silenced, stays so.

This doesn't sound like much progress, for you ought to be able to rewrite an oracle-silencing definition as an exclusion-style or penalty-style one. Maybe it's just a matter of personal taste. But we introduce another idea that seems to combine most naturally with oracle silencing. It is this: that *when* an oracle is silenced is *not* directly specified by the cryptographer creating a definition, but, instead, it is determined *automatically* by the *correctness* requirement.

Usually what's going on is that the oracle should be silenced if the adversary has asked a question where the correctness constraint dictates the response. In the real setting, that is, there's only one answer possible, as mandated by the correctness condition. The answer doesn't depend on the particular protocol $\Pi$ that underlies the game, nor does it depend on any coins. It only depends on the query history and the fact that the underlying protocol *is* correct. In such a case, we want to silence the response. See Fig. 4.

A bit more formally, one begins by defining a pair of *utopian* games $G$ and $H$. They are called "utopian" because we expect that an adversary *can* distinguish between them, just by exploiting some simple test that ought not be permitted.

The cryptographer also defines a *correctness condition* $C$, which is just the class of correct schemes. Now we automatically and generically modify $G$ and $H$, using $C$, to get a new pair of games $\overline{G}$, $\overline{H}$. The modification is to add in a little "gadget" that takes in the value that is about to be returned to the adversary and sometimes replaces it by some distinguished symbol $\perp$. The *silencing function*, which depends on $G$ and $C$, takes in the transcript that describes the interaction between the adversary and its oracles. It silences exactly when the transcript fixes the oracle's response in the real game. The silencing function (or something close to it) needs to be efficiently computable: only in this way do we know that the adversary knows that its query is inappropriate. The adversary's ability to distinguish between the silenced games is what we call the INDC measure of security: indistinguishability up to correctness.

When we carry out the paradigm described for onion encryption, the utopian games are very simple: they look like the naïve games sketched before. Each game can be described with about six lines of code. But it is not this pair of games that the adversary must distinguish. It is the silenced pair of games obtained by "compiling" the utopian games using the correctness condition earlier described.

One use of INDC security is to justify complicated games: if distinguishing them is equivalent to distinguishing the pair of games one gets from silencing utopian games using correctness, this is a demonstration that the games are in some sense right. Alternatively, INDC security can be used to conveniently define games described in no other way.

## 5 Conclusions

I want to wrap up now, making some final comments.

- First, as already mentioned, I strongly advise, when writing definitions, to completely separate the syntax from the security notion, and to pay much more attention to the former than people routinely do. Syntax guides security, and has a profound impact on what security properties we can even express.
- Another point concerns where to find problems in need of definitional work. An often overlooked source is cryptographic practice itself—cryptographic practice that hasn't met up with theory. Often things can be well established in cryptographic practice without any theoretically minded person noticing, or perhaps caring, that there is an important problem to address. Both our second and third examples, online encryption and onion encryption, are cases of theory seriously lagging practice.
- Relatedly, things that are in need of defining may be sitting in plain view, seemingly without anyone taking notice. I mentioned that there were more than a thousand papers using garbled circuits without anyone attending to the fact that what garbled circuits accomplished, on their own, hadn't been defined.
- Definitions can be wrong. When you hear somebody say "it can't be wrong, it's a *definition*", challenge this sophistry. Definitions can be wrong for a

variety of reasons, but the most important is that they fail to model what you are really wanting or needing to model. When you encounter definitions you believe to be wrong, develop your arguments and speak out, even though this can be difficult to do. It can be difficult because lots of people may be invested in the downstream work that springs from a definition, and calling a definition into question impugns everything that uses it.

- In the examples given today, and in all of the definitional work I've done over my career, I get the overwhelming sense that I am *constructing* a definition, not *discovering* it. But you hear people speak in the opposite terms, as though there are these cryptographic definitions *out there*, waiting to be discovered, and that our job as cryptographers is to find them. I don't think the world works this way. I think we create definitions in order to satisfy the needs of some particular community, and the value of a definition is the extent to which it does. The idea is explored in a separate essay [23].
- Definitional work can be practical. Many people assume the opposite. Perhaps they think that definitional work is impractical because it's slow, and practice can't wait. Definitions usually needs lots of refinement, and frequent backtracking. Nonetheless, spending the time to hammer out the definitions and make them beautiful can come to have a large practical payoff.
- Definitional work is dialectical. You might think that by working out the definitions thoughtfully enough, you will manage to arrive at the *right* definition. Being *right*, it won't *need* to evolve. It *shouldn't* evolve. But I don't buy it. My experience has been that definitional notions evolve much more than people imagine, and that the definitional process does not really terminate.
- Definitions are fundamentally fictions. We attend to the things that we want to attend to, and we ignore the rest. The rest hasn't gone away, and we do well to remember that. That definitions are fictions does not mean that we shouldn't take them seriously. We absolutely should take our definitions seriously, in the sense of wanting to fully understand their consequences, relationships, and limitations. But we should not take our definitions seriously in the sense that we delude ourselves into believing the definition is a genuine surrogate for the thing we are interested in. They are abstractions—platonic models of some aim.
- Finally, I'll comment that part of why I like working on cryptographic definitions is that it seems to involve a unique style of modelling. The creation of a definition integrates social, technical, and philosophical concerns. The last figures into the process more than outsiders might imagine; in developing a definition, philosophical discourse may even dominate discussions. Perhaps there are other disciplines where modelling has this same character, but I have yet to encounter them. In most scientific, engineering, and even social-science disciplines, empiricism plays a major role. Does the rod bend in the manner predicated by the PDE? Does the economy evolve in the way the equations predict? In making cryptographic definitions, there is rarely any experimental or observational aspect; instead, the technical work is entwined with social, philosophical, and aesthetic considerations. Ultimately, this may be the aspect of the definitional enterprise that I like most.

**Acknowledgments.** Thanks first to the coauthors of papers whose definitions I have summarized: Mihir Bellare, Viet Tung Hoang, Reza Reyhanitabar, Damian Vizár, and Yusi Zhang [4,19,25,26]. Good definitions require good coauthors. Further thanks to Tung and Yusi for helpful proofreading.

Many thanks to the NSF for their support under grants CNS 1314885 and CNS 1717542. Of course all views expressed in this paper are entirely my own.

This paper was prepared to accompany an invited talk at Latincrypt 2017, which was held in Havana, Cuba. My kind thanks to all of those involved in organizing Latincrypt and inviting my participation, particularly Program Chairs Orr Dunkelman and Tanja Lange, General Chair Luis Ramiro Piñeiro Díaz, and Steering Committee member Francisco Rodríguez-Henríquez.

Latincrypt 2017 was my first time in Cuba, a place so close to the U.S. that a woman has managed to swim that gap [21]. Yet for decades the U.S. has maintained bizarre policies towards this neighbor, causing much suffering. I myself was born in 1962, during the brief interval between the Bay of Pigs Invasion and the Cuban Missile Crisis. One might have assumed that, 55 years later, relations would surely have normalized. It is very sad that this is still not the case.

## References

1. Bellare, M., Boldyreva, A., Knudsen, L., Namprempre, C.: Online ciphers and the hash-CBC construction. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 292–309. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_18
2. Bellare, M., Hoang, V. T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: IEEE Symposium on Security and Privacy, pp. 478–492 (2013)
3. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 134–153. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_10
4. Bellare, M., Hoang, V. T., Rogaway, P.: Foundations of garbled circuits. In: ACM Conference on Computer and Communications Security (CCS 2012), pp. 784–796 (2012). Full version is Cryptology ePrint Archive, Report 2012/265 (2012)
5. Bellare, M., Hofheinz, D., Kiltz, E.: Subtleties in the definition of IND-CCA: when and how should challenge decryption be disallowed? J. Cryptol. **28**(1), 29–48 (2015)
6. Bellare, M., Kohno, Y., Namprempre, C.: Breaking and provably repairing the SSH authenticated encryption scheme: a case study of the encode-then-Encrypt-and-MAC paradigm. ACM Trans. Inf. Syst. Secur. **7**(2), 206–241 (2004)
7. Bernstein, D.: Cryptographic competitions. competitions.cr.yp.to. Accessed 1 Feb 2018
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28496-0_19
9. Boyd, C., Hale, B., Mjølsnes, S.F., Stebila, D.: From stateless to stateful: generic authentication and authenticated encryption constructions with application to TLS. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 55–71. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_4
10. Boyle, M., Salter, C.: Dual Counter Mode (2001). Unpublished manuscript. gitweb.tinyurl.com/dual-counter-mode

11. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–90 (1981)
12. Dingledine, R., Mathewson, N.: Tor protocol specification. The Tor Project. gitweb.torproject.org/torspec.git/tree/tor-spec.txt (2018)
13. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. Naval Research Lab, Technical report (2004)
14. Duong, T., Rizzo, J.: Practical padding oracle attacks. USENIX Workshop on Offensive Technologies (WOOT) (2010)
15. Fleischmann, E., Forler, C., Lucks, S.: McOE: a family of almost foolproof online authenticated encryption schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34047-5_12
16. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding routing information. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 137–150. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61996-8_37
17. Goldschlag, D., Reed, M., Syverson, P.: Onion routing. Commun. ACM **42**(2), 39–41 (1999)
18. Goldwasser, G., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)
19. Hoang, V.T., Reyhanitabar, R., Rogaway, P., Vizár, D.: Online authenticated-encryption and its nonce-reuse misuse-resistance. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 493–517. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_24
20. Netflix: Netflix/msl. github.com/Netflix/msl/wiki. Accessed 6 April 2016
21. Nyad, D.: Never, ever give up. Talk at TEDWomen 2013 event (2013)
22. Rogaway, P.: On the role of definitions in and beyond cryptography. Manuscript. web.cs.ucdavis.edu/ rogaway/papers/def.pdf
23. Rogaway, P.: Practice-oriented provable security and the social construction of cryptography. IEEE Secur. Priv. **14**(6), 10–17 (2016)
24. Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_23
25. Rogaway, P., Zhang, Y.: Onion-AE: foundations of nested encryption. In: Proceedings on Privacy Enhancing Technologies (PETS 2018), issue 2. De Gruyter Open (2018, to appear)
26. Rogaway, P., Zhang, Y.: Simplifying game-based definitions: indistinguishability up to correctness and its application to stateful AE. Manuscript (2018)
27. Syverson, P., Goldschlag, D., Reed, M.: Anonymous connections and onion routing. In: 1997 IEEE Symposium on Security and Privacy, pp. 44–54. IEEE Computer Society Press (1997)
28. Tsang, P., Solomakhin, R., Smith, S.: Authenticated Streamwise On-line Encryption. Dartmouth Computer Science Technical Report TR2009-640 (2009)
29. Vanhoef, M., Piessens, F.: Key reinstallation attacks: forcing nonce reuse in WPA2. In: ACM Conference on Computer and Communications Security (CCS 2017), pp. 1313–1328 (2017)
30. Yao, A.: How to generate and exchange secrets. In: FOCS 1986 (27th Annual Symposium on the Foundations of Computer Science), pp. 162–167. IEEE Computer Society Press (1986)
31. Yao, A.: Protocols for secure computations. In: FOCS 1982, 23rd Annual Symposium on the Foundations of Computer Science, pp. 160–164. IEEE Computer Society Press (1982)