

Alice Tutors Bob on Mature and Minimalist Cryptography

JUSTIN TROUTMAN*

June 10, 2009

Abstract

We take a concise look – with an emphasis on symmetric cryptography – at some of the issues that are responsible for why cryptography usually ends up looking bad, in practice, and fails to establish the right threat model, let alone realize it; this is largely due to a lack of cryptographic competence, and the dreaded habit of crammed-in-and-cobbled-together design. To address these issues, we, with the assistance, and comedic relief, of Alice and Bob, give several rules of thumb for sufficient and simplistic cryptographic implementations. Be prepared for a bowl of acronymous porridge, but don't worry; we'll make sure it's as easy to swallow as possible, and it might even up your Scrabble game. So, to the pulpit we go, ready to preach a sermon so desperately in need of being heard, and to which heed should be taken.

*Extorque, 114 Union Heights Blvd., Salisbury, North Carolina, 28146, USA. E-mail: justin@extorque.com

Contents

1	If you want confidentiality, you want integrity too.	3
1.1	Meet MAC	4
2	Recycling: It's good for the ecosystem, <i>and</i> your cryptosystem.	4
2.1	The Progression of Recycling	5
3	Put it together with cryptographic Elmer's.	6
3.1	Generic Composition	6
3.2	Integrity-Aware Modes of Operation	7
3.3	The Good, the Bad, and the Proof	7
3.4	Brief definitions, or "Briefinitions," if you will.	8
3.5	Juxtaposing the Two	9
4	Stop harping on key length, please!	9
5	Keep your ducks in a row and their feathers to themselves.	10
6	From Developer to Implementation to User	10
7	Conclusion	11
8	Acknowledgments	11

1 If you want confidentiality, you want integrity too.

“Hey Alice. You’re looking splendid, as usual.” “You’re not too bad yourself, Bob.” “I don’t know about you, Alice, but I’m feeling a little exposed. Why don’t we put on our matching sweaters that Vincent and Joan gave us? 100% SPN¹. How much more elegant can encryption get? The block pattern is great. Argyle just doesn’t do it for me anymore.” “You’re right, Bob. The length is just right for me, too - 128 bits. But, aren’t you forgetting something?” “We’ve got encryption. What more do we need?” “We might as well go out in our birthday paradox suits, if all we’re wearing is encryption.” “I’m not following you, Alice.” “Bob, Bob, Bob. How do we get confidentiality?” “We use encryption, Alice. Are you feeling alright?” “I’m fine, Bob, but if you don’t put on your MAC², you won’t be.” “Wait a minute. If all I want is confidentiality, why are you telling me to put on my MAC? I already checked the forecast and I don’t think I’ll need integrity today.” “You might not think you’ll need it, but you probably will. Put it on, Bob. Now.” “Okay, okay, Alice. Easy.”

Alice and Bob. At times, it’s all they can do to agree on a key, but Bob’s leading lady is going to save his ciphertext. If I were to ask you, “How do you achieve confidentiality?,” your retort might also be, “That’s easy; use encryption.” And so it goes that confidentiality is all too often the only goal that’s considered. There’s another goal though, that, if not met, results in grave consequences. For those of you who don’t know about it - you probably should; for those of you who know about it, but don’t think you need it - you’re probably wrong. That goal is integrity, and the consequences for not achieving this goal are often far worse than those for not achieving confidentiality. Now you may be thinking to yourself, “Why does he insist that we need integrity? Okay, so there are consequences for not having it, but we want confidentiality, not integrity.” The reality is quite the contrary. When we want confidentiality, the obvious thought is, “we need encryption,” but, realistically, it’s much wiser to say that if we want confidentiality, “we need authentication, too.” If that’s not enough for you, consider that one of these consequences might include losing confidentiality, as well.

In [1], Bellare cuts to the chase with his cut-and-paste attacks on IPsec, and makes a point, and very clearly at that: “*It is quite clear that encryption without integrity checking is all but useless. We strongly recommend that all systems mandate joint use of the two options*”. That was 1996. In [2], Vaudenay furthers the case for authentication with his side-channel attacks on CBC padding, with applications to IPsec. By mirroring the work of Bleichenbacher, in [3], and Manger, in [4], in the asymmetric case, Vaudenay shows that the symmetric case is just as vulnerable to these types of side channels when an adversary is able to distinguish between valid and invalid ciphertexts. In [5], Black and Urtubia extend this work, in a more efficient manner, and conclude that “*perhaps it is time to view authentication as a strongly-desirable property of any symmetric encryption scheme, including those where privacy is the only security goal*,” and that “*the way to prevent all of these attacks is to insist on integrity of ciphertexts³ in addition to semantic security⁴ as the ‘proper’ notion of privacy for symmetric encryption schemes*.” That was 2002 – six years later.

Here we are in 2008, another six years later, reviving the pioneering work of yesterday, with the “in the times” updates of today, begging, pleading, preaching, and near bullying, all for one cause. You say confidentiality is what you want – not integrity. We’re telling you that if you want the former, you better have the latter. In short, if you don’t want it – tough. Use it anyway!

¹Substitution-Permutation Network.

²Message Authentication Code.

³See §4.

⁴See §4.

1.1 Meet MAC

Meet MAC, otherwise known by its full name, *Message Authentication Code*. If you've already met, it won't hurt to get to know each other a little better. Laymanizing this as much as possible, a MAC achieves the goal of integrity; if an adversary attempts to manipulate ciphertext, a MAC detects it. You feed the MAC a shared secret key and a message of arbitrary length, and it spits out a *tag*. Let's say Alice is sending a message to Bob. She'll encrypt it, then compute a tag on the ciphertext⁵[6, 7], of which it will accompany on its way to Bob. When Bob receives the message, he recomputes a tag on the ciphertext, using the same shared secret key, and compares it with the tag that accompanied the ciphertext. If the tags match, the integrity of the message has been preserved; if not, the ciphertext has probably been tampered with.

When we're ready to put this wisdom into action, we recommend CMAC⁶ for message authentication schemes; it's a MAC that uses a block cipher as its underlying primitive. This allows us to use the AES⁷, which we already like to use for message encryption. Why do we like the AES and why do we want to use it for both encryption (i.e., to achieve confidentiality) and authentication (i.e., to achieve integrity)? Well, go pour yourself a glass of your poison-of-choice⁸, then sit back down. There's more crypto-sorcery to come.

2 Recycling: It's good for the ecosystem, *and* your cryptosystem.

With a cryptographic palette full of block ciphers to choose from, you might find yourself wondering which to use. The answer is simpler than you might think: Use the AES[8, 9], unless there are constraints that don't allow you to. (A possible exception would be niche applications that impose environmental constraints for which another block cipher is more suitable.) The most compelling reason for using the AES is based on what happens as a result of being a standard - it receives more cryptanalytical attention than any of the other AES finalist candidates, which gives us a solid cryptographic position on recommending it. Not only that, but it fits in with the engineering principle of "recycling primitives." Oh, and we don't want to hear any of that "I need other block ciphers just in case the AES is broken" sputter; the odds that you'll make an implementation blunder are a stack of magnitudes greater than the surfacing of a practical attack on the AES. So, here's something we ask of you: **Unless you absolutely cannot, use the AES.** "What's so holy about the alliance between the AES and recycling primitives?," you might question. Alice, you take it from here.

"Hey Alice, I picked up some things at the Diffie Mart." "Please tell me you didn't forget the Hellman's mayonnaise." "Calm down, Alice. It's right here." "Whew. So, what else did you pick up?" "Well, they had a sale on fresh Twofish and Blowfish, so I got a filet of each. Let's see, I bought a box of HMAC⁹ 'n cheese - the SHArp¹⁰ cheddar kind. Oh, and while I was in the check-out lane, I saw a

⁵For cryptographic reasons, as examined in [6, 7], we recommend computing a tag on the ciphertext, instead of the plaintext.

⁶Block cipher-based Message Authentication Code.

⁷Advanced Encryption Standard.

⁸Ours happens to be sweet iced tea.

⁹Hash function-based Message Authentication Code.

¹⁰Secure Hash Algorithm.

MARS¹¹ bar. I know, I know, but I've been craving one." "Bob, why do we need all of that?" "The more, the merrier, Alice! The more, the merrier!" "Oh Bob, now we have to worry about cooking fish, and you **know** I can't cook fish. HMAC 'n cheese and a MARS bar? Seriously, Bob. Seriously. Things would be a lot easier on our wallets, and our stomachs, if we just stick with our AES diet. It has all we need, and there's a lot of analysis to back it." "Can I just keep the MARS bar?" "Bob." "Yeah, yeah, Alice. You win."

Bob is a complete smörgåsbord addict. To him, more is more, and more is good. What he's missing is the fact that cryptographic implementation isn't served well by this way of thinking. Academia-borne cryptographic design has a remarkably good-looking track record, and you'd hope that this would carry over to its implementation. Tragically, this is far from the case. In practice, when cryptography fails, it's almost never the cryptography's fault; if we're looking to point fingers, we usually look in the direction of the implementation. Many developers, like Bob, try to cram in as much cryptography as possible. Oh yeah. Let's throw in every block cipher known to Bruce. How about an assortment of stream ciphers and hash functions? How many do you intend to invite to this primitive party? If you're a "Bob," you might think something along the lines of, "Aren't more options better?" If you're an "Alice," you'll see that more options lead to more complexity.

The more you attempt to shove into an implementation, the more likely you are to make a mistake. And believe me when I say that even the most subtle of mistakes can pack a huge punch. Don't get caught up in the illusion that you need a lot to do a lot; in fact, you can do a lot with a little. Less is more. Be resourceful. Reuse what's already there; that is, *recycle*. If you can use a single primitive for multiple purposes, then by all means, do so. Simplify the implementation by using good design rationale, because the fruits of your decisions are passed on to the implementation and the user. Don't burden the implementation with unnecessary complexities or smother the user with a plethora of configurations, as this is where things usually go wrong.

By using the AES as the underlying primitive in our encryption scheme, authentication scheme, and PRF¹², or *pseudorandom function*, we do our implementation both cryptographic and engineering favors. Two birds, one - oh, you know¹³. You see, by recycling primitives, we not only simplify the implementation, but we take advantage of the cryptanalysis that the AES has received, in each of the schemes in which we recycle it. This reflects the perpetually mutual relationship between cryptography and engineering, within the context of recycling primitives, when the correctness and security of the implementation are at stake. It's great to know what should be in your toolbox, but that doesn't amount to much if you don't know how to use it; this is why you're probably about to say, "So, guys, we know we need confidentiality and integrity, and we know we can use the AES to achieve both, but how do we put this all together?" Time to break out the alphabet adhesive.

2.1 The Progression of Recycling

We've shown you that recycling is the mutual tie that binds good cryptographic design and cryptographic engineering, but you might not realize to what extent it does so. Recycling is a paradigm that can be applied from conception to cellophane. It goes like this. In the design of the components of primitives, recycle design strategies (e.g., the wide trail strategy[9, 10, 11], as exhibited by Rijndael).

¹¹IBM's submission to the AES selection process.

¹²Pseudorandom function; the MACs we discuss in this essay, namely the identical OMAC1 and CMAC constructions, are good PRFs, which implies that they're also good MACs.

¹³No birds were harmed during the writing of this essay.

In the design of primitives, recycle components from other primitives (e.g., the S-boxes of Rijndael). In cryptographic implementations, recycle primitives (e.g., use the AES for encryption, MAC, and PRF schemes). What begins as a conservative way to design cryptography turns into a conservative way to implement it, which is certainly a good thing, from design to deployment. As a cryptosystem-friendly school of thought, we affectionately refer to this as *green cryptography*.

3 Put it together with cryptographic Elmer’s.

Now for the bowl of acronymous porridge. As promised, though, it won’t be so bad. Without any ado, we’re going to dive right in, by telling you what you need; that is, IND-CCA2 \wedge INT-CTXT security[7]. (Read \wedge as “and.”) IND-CCA2 and INT-CTXT’s full names – *Indistinguishability under adaptive Chosen-Ciphertext Attack* and *Integrity of Ciphertexts*, respectively – are definite party starters. In an adaptive chosen-ciphertext attack, an adversary is given access to a decryption oracle,

Fortunately, we’re not going to require that you know the theory behind this; instead, we’re going to show you the two ways you can go about achieving it: *generic composition* and *integrity-aware modes of operation*. We’re ready if you are.

3.1 Generic Composition

You take an encryption scheme and an authentication scheme, and apply them in a certain order. The *Encrypt-then-Authenticate*, or *EtA*, composition is that certain order, and a sensible recommendation for authenticated encryption schemes. Why? Well, not only does it afford us the ability to achieve IND-CCA2 \wedge INT-CTXT security, but it’s also secure from all angles, making it less error-prone[6, 7]; needless to say, anything that makes cryptography easier to get right is a Good Thing™. That, in a nutshell, is a generic composition.

Generically speaking, apply an IND-CPA (*Indistinguishability under Chosen-Plaintext Attack*) secure encryption scheme and a SUF-CMA (*Strongly Unforgeable under Chosen-Message Attack*) MAC, in this composition. More specifically, for instance, encrypt first with AES-CTR[12, 13], then, compute a MAC on the ciphertext, using CMAC-AES[14], using separate keys for the encryption and authentication. For you developers itching for pseudo code, or something along those lines, let’s move to the algorithm. (Get it? Algorithm. Rhythm. Right.)

Simply put, we’re dealing with two base schemes: Message Encryption ($\mathcal{ME} = (\mathcal{K}_e, \mathcal{E}, \mathcal{D})$) and Message Authentication ($\mathcal{MA} = (\mathcal{K}_a, \mathcal{T}, \mathcal{V})$). $\overline{\mathcal{ME}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ is the authenticated encryption scheme obtained by composing $\overline{\mathcal{ME}}$ and $\overline{\mathcal{MA}}$ in the *Encrypt-then-Authenticate* composition¹⁴.

¹⁴In [7], the authors refer to this as “Encrypt-then-MAC”, which is equivalent to our “Encrypt-then-Authenticate.” We made this alteration in order to better coincide with our base schemes of Message *Encryption*, \mathcal{ME} , and Message *Authentication*, \mathcal{MA} ; as such, their notation for Symmetric Encryption, \mathcal{SE} , and the MAC key, \mathcal{K}_m , becomes Message Encryption, \mathcal{ME} , and authentication key, \mathcal{K}_a , respectively. However, there is no change in the meaning of the notation, so please refer to [7] for a detailed look at this composition and its associated security proofs.

Algorithm $\bar{\mathcal{K}}$ $K_e \xleftarrow{\$} \mathcal{K}_e$ $K_a \xleftarrow{\$} \mathcal{K}_a$ Return $K_e K_a$	Algorithm $\bar{\mathcal{E}}(K_e K_a, M)$ $C' \xleftarrow{\$} \mathcal{E}(K_e, M)$ $\tau' \xleftarrow{\$} \mathcal{T}(K_a, C')$ $C \leftarrow C' \tau'$ Return C	Algorithm $\bar{\mathcal{D}}(K_e K_a, C)$ Parse C as $C' \tau'$ $M \leftarrow \mathcal{D}(K_e, C')$ $v \leftarrow \mathcal{V}(K_a, C', \tau')$ If $v = 1$ then return M else return \perp .
---	---	--

The premise is simple: Key generation algorithm, \mathcal{K} , is used to derive both encryption key, K_e , and authentication key, K_a . Alice encrypts a plaintext message, M , using encryption algorithm, \mathcal{E} , with encryption key, K_e , yielding the ciphertext, C' . She then computes a tag, τ' , on the ciphertext, C' , using tagging algorithm, \mathcal{T} , with authentication key K_a . She appends the tag, τ' , to the ciphertext, C' , yielding the tagged ciphertext, C , which she sends to Bob. Bob parses C as $C' || \tau'$. He then verifies the ciphertext, C' , using verification algorithm, \mathcal{V} , with authentication key, K_a . If the tag he computes matches the tag, τ' , that Alice sent along with the ciphertext, C' , he proceeds to decrypt the ciphertext, C' , using decryption algorithm \mathcal{D} , with encryption key, K_e , thus returning the plaintext message, M (i.e., if $v = 1$ then return M , signifying a valid ciphertext); if it doesn't match, the ciphertext is discarded (i.e., else return \perp , signifying an invalid ciphertext).

3.2 Integrity-Aware Modes of Operation

What if we had a single, self-contained mode of operation, that addressed both confidentiality and integrity? You're in luck; we do. Confidentiality modes of operation, such as CTR¹⁵, are, well, confidentiality-only; they do nothing for integrity. With an integrity-aware mode of operation, you're getting integrity preservation on top of confidentiality preservation. Two birds, one – there's really no need to call PETA¹⁶. Oh, and such modes are often designed to directly achieve $\text{IND-CCA2} \wedge \text{INT-CTXT}$ security. A prime example of this is EAX[15], a mode for *authenticated-encryption with associated data*, where *associated data* refers to information that you wouldn't encrypt, such as a packet header. It's a two-pass scheme, meaning that encryption is taken care of in the first pass, while authentication is handled by the second pass.

What's particularly nifty about EAX is that, if you were to peer below its epidermis, you'd see a generic composition, known as EAX2; in fact, it uses CTR mode for confidentiality and OMAC1¹⁷ for integrity, with the latter being equivalent to CMAC. While a generic composition uses two separate keys – one for encryption and one for authentication – EAX2 collapses two keys into one. If you're partial towards generic composition, you really can't go wrong with EAX2; if you're in the market for an integrity-aware mode of operation, you really can't go wrong with EAX, either, given that EAX2 makes up its guts. As far as authenticated encryption is concerned, cryptographers have ensured that you're well covered, and efficiently so. We're taking out more birds than even Hitchcock's imagination could summon.

3.3 The Good, the Bad, and the Proof

CTR, OMAC1, CMAC, and EAX carry proofs of security that depend on the underlying block cipher being a secure PRP, or *pseudorandom permutation*. Furthermore, they're components in generic

¹⁵Counter mode.

¹⁶Refer to ³ for reassurance. Cryptographer's honor.

¹⁷One-key Message Authentication Code.

compositions and integrity-aware modes of operation that are proven to be secure in the sense of $\text{IND-CCA2} \wedge \text{INT-CTXT}$. But, we can't toss in a term like "proof," without sprinkling in a little on the heated debates hanging heavily around it, like gold around Mr. T. Unfortunately, "provable security" has been juggled a bit too carelessly by some and misunderstood by others, so the opposition's sentiment[16] is understood. They're not absolute proofs of anything, so don't stretch them that far. Obviously, there's a need to better gauge the balance between our expectations and their limitations.

Despite all of the opinionated back-and-forth, when the dust settles, we're of the mindset that provable security is useful cryptographic design metric. Security proofs are a good "sanity check," if you will. They make it easier to hone in on potential design weaknesses, which, believe you me, can save you from the most brain-jangling of cryptographic migraines. **Yes!** That's right! Real systems can fail[1, 5] for not taking adaptive chosen-ciphertext attacks into consideration! Proper integrity preservation, through the use of a MAC (Aha!) or integrity-aware mode (Aha Part Deux!), address these attacks.

One could, and probably should, write a book about provable security and its cans and cannots, but we'll leave you with this to consider: "Cleanliness" is a desirable attribute for making analysis easier, from design to deployment, and provable security is a proponent of that. It's easier to design a good system if you can clearly define your goals and assumptions, and justify your design claim, such that if the assumptions hold, your system meets the goals. Ad hoc approaches won't pass the muster in this department. Now, place your bets. We know where ours will be.

3.4 Brief definitions, or "Briefinitions," if you will.

We've thrown several attack model acronyms at you, without any definitions, thus far, and with good reason: We want to spare you from the math. In the interest of not keeping you completely in the dark, here are some brief definitions, or "briefinitions," if you will. If you're looking for directions to the Carnival de Matemática¹⁸, consult the references we've provided – especially the work of Bellare and Namprempe, in [7].

In a chosen-plaintext attack (CPA), the adversary has access to an encryption oracle, which he will query with arbitrary plaintexts to be encrypted, with the corresponding ciphertexts being returned; the adversary is allowed to make multiple, adaptive queries based on information gained from previous encryptions. IND-CPA security means that the adversary is not able¹⁹ to distinguish between the encryptions of different messages, even if he has the ability to make arbitrary encryptions.

In an adaptive chosen-ciphertext attack (CCA2), the adversary also has access to a decryption oracle, which he will query with arbitrary ciphertexts to be decrypted, with the corresponding plaintexts being returned; the adversary is allowed to make multiple, adaptive queries based on information gained from previous encryptions or decryptions. IND-CCA2 security means that the adversary is not able to distinguish between the encryptions of different messages, even if he has the ability to make arbitrary encryptions and decryptions.

To have integrity of ciphertexts (INT-CTXT), the adversary must not be able to produce a ciphertext that was not previously produced by the sender, regardless of whether or not the underlying plaintext is "new." The adversary is allowed to mount a chosen-message attack. For a MAC to be strongly unforgeable under a chosen-message attack (SUF-CMA), the adversary must not be able to

¹⁸When in need of spice, reference Brazil.

¹⁹Read "is not able" as "computationally infeasible."

find a new message-tag pair, even after a chosen-message attack. Note that any PRF is a SUF-CMA MAC.

To recap, when considering authenticated encryption schemes, it's common to think of it in terms of coupling an integrity notion, like INT-CTXT, with a confidentiality notion, like IND-CPA. Let's suppose our authentication scheme meets the criteria for INT-CTXT \wedge IND-CPA. INT-CTXT \wedge IND-CPA \rightarrow IND-CCA2 (Read \rightarrow as "implies."), which means that our authenticated encryption scheme is IND-CCA2 \wedge INT-CTXT secure – exactly what we want. Apply a SUF-CMA MAC, like CMAC-AES, to the ciphertext of an IND-CPA secure encryption scheme, like AES-CTR, or, use an integrity-aware mode built to achieve IND-CCA2 \wedge INT-CTXT security directly, and you get just that.

3.5 Juxtaposing the Two

In a generic composition, the encryption and authentication schemes are separated, and applied in an arbitrary order; in our case, we've chosen to encrypt first, then authenticate. As these schemes handle two different conceptual goals – confidentiality and integrity – it makes sense, aesthetically, to apply them separately; in turn, this makes the correctness of the implementation more clearly defined, while making it more flexible, as well. With an integrity-aware mode, encryption and authentication are wed together into a single mode, which requires a lot less from the developer, in regards to thinking about encryption and authenticate separately; instead, the developer thinks about one mode that does it all. This could very well makes mistakes less likely. Aside from these cryptographic and engineering observations, there are performance attributes to take into consideration. One of the most obvious of these attributes is exemplified through EAX, which, for example, collapses two keys into one, while a generic composition requires two keys; this saves on key material and key-setup. When all is said and done, you can get IND-CCA2 \wedge INT-CTXT security with either. You can do either securely, so don't be afraid to consider the attributes of whichever may be especially well-suited to your environment.

4 Stop harping on key length, please!

The constructions we've discussed so far, as well as the primitives on which they are based, are keyed, so it's only right that we look at the hype surrounding key length. Whew. Where do I begin? Despite being one of the easiest cryptographic decisions you'll make, developers often dwell on key length as if it's a religious sacrament. Security products, these days, are indicative of this obsession. With no reservations, here's The Bottom Line™ is: Choose a key length and move on. Forgive us, in advance, for this cliché nipping at our heels, but it really form fits this situation; that is, "it's not the length that matters, but how you use what you've got." Key lengths and security levels are two different things. In other words, considering that 128 bits of security is what we generally aim for, you could argue that a 256-bit is a conservative way to go about it. However, if you're going to use 256-bit keys, then do so for the right reason. Don't do it because you think the AES with a 128-bit key is insufficient; it's likely more than sufficient. Do it because you understand that key material often leaks in the real world, and a larger key compensates for this[17].

5 Keep your ducks in a row and their feathers to themselves.

Although this isn't crypto-centric, but a general gem of wisdom, it'd be irresponsible not to touch on this. Cryptography shares its environment with others – often in rather tight quarters. What we call “keeping your ducks in a row and their feathers to themselves” is a paradigm that's founded on *modularity*. In a tightly integrated system, running rampant with dependencies, it's not unlikely for a failure to be global, in that a failure in one place causes a failure in another. In a modular design, failure becomes local, as it should be. If something goes wrong, it does so within the confines of that module. Of course, you can't always get rid of dependencies, but you can minimize the extent to which they exist. Maintain the simplicity of how modules interface with each other. Make it easy to analyze module behavior. The correctness and security of an implementation is magnified through modularity; it aims to keep things clean, which is a Godsend-of-an-attribute, come analysis time.

6 From Developer to Implementation to User

With implementations like loose slacks, we've got a belt, yet we can't seem to put it on right. Developers don't need to spend so much time worrying about which block cipher and key length to use; this is likely to be the easiest part of your decision-making process. Cryptography is like that good student in class, always doing what you ask of it, and doing it well. In other words, your attention is direly needed elsewhere; rest assured that cryptography won't be the first to misbehave. Sear this into long-term memory: Cryptography is only as effective as the implementation allows it to be, so the logical path to better odds in getting it right is to simplify the way developers look at it. It follows that the implementation, and essentially, the user, are at the mercy of the developer's decisions. If the developers aren't given the heads up, things start to look pretty bleak as you travel from developer to implementation to user.

If you get it right, cryptography will be good to you. We'll go so far as to say that it's likely the most dependable part of your system. (If cryptography is the weakest part of your system, you're doing incredibly well. Teach us.) While we tout the simplification of cryptographic design decisions, cryptography isn't paint-by-numbers simple; it can be downright difficult. So, if you take anything from this essay: *Don't do cryptography without a cryptographer on board!* Be that the exhortation of our careers. There are some things that we can't do, so we call a professional. For instance, you wouldn't perform a root canal on yourself; you'd trust this procedure to a dentist. (At least, we hope you would; if you've ever done this yourself, you're probably Chuck Norris or his offspring.)

On the other hand, there are some things we could probably learn to do ourselves, with a little time and patience, yet we *still* often call a professional. Take painting a room or installing ceramic floor tile, for example. What are two DIY projects are more easily outsourced. Besides, by having an experienced someone do the job, you reduce the likelihood of mistakes happening, which, as you know by now, is exactly what cryptographic implementation counts on – reduced likelihood of mistakes. Why is it that folks trust professionals to do jobs, both PhD-only and piece-of-cake, yet they chance the fantasy that they can take on the intricately artistic and scientific nooks and crannies of cryptographic design?

Although we'd like to tell you that educating developers on how to properly implement cryptography will solve the issue, this is where the bad news comes in. It's not enough that developers know how to use the tools properly. First, they need to know which tools they should be using; this is what

we know as threat modeling. It's where a good design begins, so you'd better get it right, because everything thereafter depends on it. Look at this as the digital analog of a physical structure's foundation; everything else will collapse under the weight of a failure to threat model properly. Risking this without know-how is risking it all.

While the correctness and security of an implementation dictates whether or not the cryptography can do its job, the threat model dictates which cryptography is right for the job, and it requires what a cryptographer's forte delivers. We can educate developers to a certain extent, but, after all, reaching security is first recognizing insecurity. Knowing what to do requires knowing what **not** to do. And, let us be the first to tell you - this takes a certain kind of expertise that's born of experience, experience, and more experience. There's no substitute for having a cryptographer on hand.

The reality is that we don't need better cryptography; we need better implementations. The more we simplify the way developers approach cryptography, the simpler, and better, the implementations will be. There's an enormous gap between cryptographers and developers, and education is one of the ways we can lessen it. Whaddya say? We're game if you are.

7 Conclusion

There really is no conclusion to this matter. The commercial arena of cryptography is the poster child of what happens when cryptography is done without a cryptographer in sight. Educating developers is a solid effort in mitigating the potential hazards of implementing cryptography, but it's far from a panacea; if anything, it's a small, but worthwhile, piece of weaponry in this "war on incompetence" (don't mind the parallel of political satire). Bear in mind that much of the cryptographic software you see is the product of software vendors. Not cryptographers. Not even security folks. Without a cryptographer close by, or a well-versed-in-cryptography security expert, at least, you're placing your bets on software vendors to establish the right threat model; it's a crapshoot, really, and the odds aren't in favor of the consumer.

While we only skim the surface of what needs to be considered, this essay will have done its job if it steers developers towards more fruitful cryptography, through mature and minimaist design. We hope that you'll add these tidbits to your arsenal of design philosophies, and wield them liberally, the next time you're faced with implementing things cryptographic. It's about time that the instantiation of cryptography reflects the decorated heritage that decades of academic research have begotten. On behalf of Alice and Bob, we bid you farewell and better cryptography. Oh, and allow us to stress, once more, a point so dear to our cryptographic hearts, in a Ray Parker Jr. slash Ghostbusteresque tone: *Who you gonna call? Cryptographers!* We ain't 'fraid of no crypto.²⁰

8 Acknowledgments

We extend our utmost gratitude to David Wagner, Chanathip Namprempre, Bruce Schneier, Paulo Barreto, Vincent Rijmen, Eli Biham, and Peter Gutmann, for their indispensable insight and much-appreciated assistance in bettering the way developers look at cryptographic engineering. We also

²⁰And so it goes that we're not paid for our pun.

appreciate the work of Daniel Day-Lewis, who taught us that there is no greater shame to cast upon someone than to drink their milkshake. Be this our attempt to keep that from happening to you.

References

- [1] S. M. Bellare, “Problem areas for the IP security protocols,” in *SSYM’96: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, (Berkeley, CA, USA), pp. 1–16, USENIX Association, 1996.
- [2] S. Vaudenay, “Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ...,” in *EUROCRYPT ’02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, (London, UK), pp. 534–546, Springer-Verlag, 2002.
- [3] D. Bleichenbacher, “Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1,” in *CRYPTO ’98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, (London, UK), pp. 1–12, Springer-Verlag, 1998.
- [4] J. Manger, “A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0,” in *CRYPTO ’01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, (London, UK), pp. 230–238, Springer-Verlag, 2001.
- [5] J. Black and H. Urtubia, “Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption,” in *Proceedings of the 11th USENIX Security Symposium*, (Berkeley, CA, USA), pp. 327–338, USENIX Association, 2002.
- [6] H. Krawczyk, “The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?),” in *CRYPTO ’01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, (London, UK), pp. 310–331, Springer-Verlag, 2001.
- [7] M. Bellare and C. Namprempre, “Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm,” in *ASIACRYPT ’00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, (London, UK), pp. 531–545, Springer-Verlag, 2000.
- [8] “Specification for the Advanced Encryption Standard (AES).” FIPS Publication 197, 2001.
- [9] J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.
- [10] J. Daemen and V. Rijmen, “The Wide Trail Design Strategy,” in *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, (London, UK), pp. 222–238, Springer-Verlag, 2001.
- [11] J. Daemen and V. Rijmen, “Security of a Wide Trail Design,” in *INDOCRYPT ’02: Proceedings of the Third International Conference on Cryptology*, (London, UK), pp. 1–11, Springer-Verlag, 2002.
- [12] H. Lipmaa, P. Rogaway, and D. Wagner, “Comments to NIST Concerning AES-modes of Operations: CTR-mode Encryption,” in *Symmetric Key Block Cipher Modes of Operation Workshop*, (Baltimore, Maryland, USA), 2000.

- [13] M. Dworkin, “Recommendation for Block Cipher Modes of Operation: Methods and Techniques.” NIST Special Publication 800-38A, 2001.
- [14] M. Dworkin, “Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication.” NIST Special Publication 800-38B, 2005.
- [15] M. Bellare, P. Rogaway, and D. Wagner, “The EAX mode of operation,” in *Fast Software Encryption (FSE) 2004*, (Berlin, Germany), pp. 389–487, Springer-Verlag, 2004.
- [16] B. Schneier, “Mathematicians vs. Cryptographers.” Schneier on Security (2007/09 Archive), 2007.
- [17] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. New York, NY, USA: John Wiley & Sons, Inc., 2001.