Mackerel: A Progressive School of Cryptographic Thought

JUSTIN TROUTMAN AND VINCENT RIJMEN

Justin Troutman is a cryptographer with research interests in designing authenticated encryption constructions, optimizing the user experience of cryptographic products, and analyzing methodologies for signals intelligence. He has worked with Microsoft, Google, Duke University, and IEEE. He co-developed the "green cryptography" strategy for minimizing implementation complexity, while maximizing cryptographic security, by recycling components for multiple purposes. He is currently organizing international workshops that explore topics such as optimizing the effectiveness of cryptography for journalists, and the intersection between real-world cryptographic design and experience design.
justin@justintroutman.com

Vincent Rijmen is professor with the Department of Electrical Engineering at the University of Leuven and the security department of the research institute iMinds (Belgium). He is co-designer of the algorithm Rijndael, which has become the Advanced Encryption Standard (AES), standardized by the US National Institute for Standards and Technology (NIST) and used worldwide, e.g., in IPSec, SSL/TLS, and other IT-security standards. Rijmen is also co-designer of the WHIRLPOOL cryptographic hash function, which is standardized in ISO/IEC 10118-3. Recent research interests include security of hash functions, design of methods for secure hardware implementations and novel applications of computer security.
vincent.rijmen@esat.kuleuven.be

Troutman and Rijmen offer a framework for creating and fielding new security systems involving cryptography. Without this or a similar framework to coordinate efforts between different stakeholder communities, we are doomed to continue providing systems that provide less than expected, take too long from conceptual birth to fielding, and still leave us at unacceptable risk.

Their framework is "chunked" to nicely fit the natural flow of effort from cryptographers to developers to users; this allows each profession to maximize its strengths and not be unnecessarily befuddled by work going on in sister professions that are also needed to complete full delivery of effective systems—systems that can be deployed and safely used by customers with minimal training.

Is this work perfect and fully fleshed out? NO. Is it a big step in the right direction? I think YES! Read and enjoy; it is an easy read that should plant interesting concepts in your mind that will take root and grow, leading to further needed developments.

—Brian Snow, former Technical Director of Information Assurance at NSA.

Cryptography is hard, but it's the easy part. It's an entanglement of algorithms and assumptions that only a cryptographer would find poetic, and we're at a point where strong cryptography is arguably the most robust aspect of a system's security and privacy posture. To a consumer, however, cryptography is still an esoteric sort of black magic whose benefits are out of reach. Developers: If you feel we've dropped the ball on safely implementing cryptography—which we have, and horribly so—this doesn't hold a candle to how pitifully we've failed at interfacing the benefits of cryptography to consumers. Our contribution to potentially solving this problem, dubbed Mackerel, is a design and development framework for developers that's based on the premise that *real-world cryptography is not about cryptography; it's about products.*

First, let's look at a process that works, and with which most of us are familiar: buying and driving an automobile. You decide it's time to buy a new vehicle, so you drive to the nearest car lot of your choice. You're greeted by a friendly salesman who wants nothing more than to put you in a new car that day. He needs to sell and you need to buy, so today might be a double-win for both of you. You tell him that with today's gas prices, you need something that gets good mileage, but that you also need something with decent towing capabilities, since you pull a camper to your favorite campground in the mountains. Oh, and with three kids and in-laws, you need a third row of seating. Using his oracle-like knowledge of vehicle statistics, the salesman walks you over to a sporty, yet eco-friendly, SUV that strikes the right balance for all your requirements. You feel the leather seats, admire the hands-free navigation system, and even take it for a test drive. A credit check and some paperwork later, and you're pulling out of the lot in your brand new set of wheels.

When you sit in the car, you shut the door, and (hopefully) buckle up, then proceed to insert the key into the ignition switch, turn it, and the process of internal combustion automagically happens before you. You shift into the appropriate gear, press your foot on the accelerator pedal, and you're off. At no point did you have to understand the mechanics of the vehicle or the process of internal combustion; you simply had to insert a key, turn it, shift a knob, and step on a pedal. In front of you are several indicator lights that give you visual and aural cues

that something needs attention. It lets you know if you're about to run out of gas, need an oil change, or if it's something that you should probably have a mechanic check out ("check engine"). You're able to thoroughly enjoy and benefit from the wonder of the automobile, without understanding the physics or mechanics; at the most, your experience as a user involves limited engagement with an intuitive user interface.

So, how can the cryptographic process learn from the consumer automobile experience? Well, we've stated that in order to properly realize the benefits of cryptography as a product, we need to employ the right process—one that respects the roles of people involved. These aforementioned people make up three groups: cryptographers, developers, and consumers. The first mistake, and the cardinal sin, is trying to get everyone on the same page. It's a true exercise in futility because cryptography looks different as it flows from cryptographer to developer to consumer. They each assume distinct roles that require different types of expertise. Ideally, what we want is a process that respects these roles and doesn't ask them to make decisions outside of their realm of expertise. Tragically, it rarely ever happens this way, and we devolve into a modern-day Tower of Babel, trying to collectively build something without having a clue as to what the other is saying. To remedy this, it's paramount that we notice the two relationships that exist here—cryptographer-to-developer and developer-to-consumer—where keeping a tight bond between the former is necessary for *underlying implementation assurance* (think mature and minimalist API), while doing so for the latter is necessary for *user interface accessibility* (think tactile and palatable GUI).

Cryptographer-to-Developer Relationship

Let's start with the cryptographer-to-developer relationship. Cryptographers need to approach developers with a particular golden rule in mind: *cryptographic implementations usually fall apart at the implementation level, not at the cryptographic level*. What this really means is that cryptographers need to create and promote a more benign surface for developers. It's not just about making it easy to get things right; it's even more about making things hard to get wrong. One way to achieve that is through what we call "green cryptography" [1, 2, 3] (extended drafts at justintroutman.com), which calls for the recycling of mature and minimalist components whenever and wherever it makes sense; for example, you can do authenticated encryption (and you should always be doing both authentication and encryption) with a single primitive, like the Advanced Encryption Standard (AES), by using Counter mode (CTR) for encryption and Cipher-based Message Authentication Code (CMAC) for authentication. Or even easier to implement would be an Authenticated Encryption with Associated Data mode (AEAD), which handles both encryption and authentication without the need for two separate modes. EAX (Encryption and Authentication with Associated Data), for example, is essentially a combination of CTR and One-key Cipher Block Chaining Message Authentication Code (OMAC1; equivalent to CMAC), but doesn't require that you manually combine CTR and CMAC; EAX kills two figurative birds with one stone. Not only that, but this particular construction gives you two of the strongest notions of confidentiality and integrity that we have: indistinguishability against Adaptive Chosen-Ciphertext Attacks (IND-CCA2) and Integrity of Ciphertexts (INT-CTXT). Here's a memo you should never say you didn't get: the order of encryption and authentication matters, and it follows that encrypting the plaintext, first, then authenticating the resulting ciphertext, second, is the easiest to get right, hardest to get wrong, and comes with the tightest notions of confidentiality and integrity.

There have also been recent attempts to build cryptographic APIs for developers that make things easier for developers to *safely* implement, such as Keyczar [4] from Google's security team. It achieves this safety by choosing secure default parameters (e.g., block ciphers and key lengths), and automatically taking care of consequential things such as key rotation and IV generation; this is that "benign" surface we mentioned earlier. And speaking of implementation failure as the likely center of catastrophe, there's a class of attacks that preys on the actual software and hardware implementations of cryptography, dubbed "side-channel attacks," in which everything from timing differences to power fluctuations can leak information about plaintext and keys. Fortunately, there's a library with side-channel attack resistance in mind called NaCl (a reference to cryptographic "salt"); with NaCl [5], although you can use standards such as the AES, you have the option of using Daniel J. Bernstein's own cryptographic primitives, such as the fast stream cipher Salsa20 [6], for encryption; there's also the secure message authentication code (MAC), called Poly1305-AES [7], which, although specified for the AES, can be used with other drop-in replacement ciphers. Keyczar and NaCl are important steps toward safer implementations, but they are far from ideal and represent an inch in the miles we need to go. Only by strengthening the relationship between cryptographers and developers can we get there.

Developer-to-Consumer Relationship

Now, let's tackle the developer-to-consumer relationship. Cryptographic software is the quintessential martyr of usability deprivation, a Rube Goldbergian gauntlet of epic distortion. (For our readers from the UK, "Heath Robinsonian." In fact, that's probably the most appropriate name to use, given the cryptographic context.) In [8], they capture most of the reasons why PGP (Pretty Good Privacy) and, by extension, its open source cousin, GPG (GNU Privacy Guard) share the role as poster children for tremendously useful ideas that, although used fervently by some, elude the majority of consumers because of their lack

of tactility and palatability and by asking consumers to make configuration decisions far outside their expertise. To be fair, PGP was as novel as it was timely, because at that instant, back when cryptography was a munition, we finally had something that didn't previously exist: a way to keep our email conversations secure and private, with strong cryptography. The point is that it predated the era of usable security and privacy research, and to this day, we still haven't improved much on making it easy to benefit from cryptography. Having said that, we have made strides in recent years when it comes to mediating the marriage of usability with security and privacy tools; in fact, there are academic laboratories focused on it (e.g., Carnegie Mellon's CyLab Usable Privacy and Security Lab, or CUPS) and conferences dedicated to it (e.g., Symposium on Usable Privacy and Security, or SOUPS). These are pioneering efforts that must exist, and we're better for them; on the other hand, cryptography is such a niche subset of security and privacy, and the focus of only a minute portion of this research.

In actuality, to channel [9], "usable cryptography" is as much of an oxymoron as it is manifest destiny; in fact, it's the benefits of cryptography that we should strive for as manifest destiny. Cryptography, itself, as a usable thing, doesn't exist; the utility of cryptography and the usability of a product that implements cryptography exist on entirely different planes. "Usable cryptography" is akin to saying "usable internal combustion." Consumers don't want internal combustion; they want to drive. Just like internal combustion, cryptography is an implementation detail that shouldn't be exposed to the consumer. That's right, consumers rarely, if ever, want cryptography directly; they need what it provides, but that's an entirely different problem. What the consumer actually wants is a useful product, where usefulness ("what am I getting out of this?") is determined by utility ("what does it do?") and usability ("how easily can I do it?").

To exhibit Mackerel as a philosophy for guiding product design, imagine that you're a journalist working under turbulent conditions, in an oppressive environment, and you need to communicate securely and privately with your source; you need an app for your smartphone, and such an app must optimize tactility and palatability, by focusing on: (1) zero learning curve (works with little to no training), (2) rapid-fire accessibility (works intuitively and like the apps you're used to), and (3) minimal code footprints (to simplify, and encourage, third-party auditing). A high-level API could be used to abstract away low-level components, while being conscious of side-channel attacks. Such an API could rest inside of a tactile and palatable GUI that caters to the desires of the user, without exposing you to the complex internals. Ultimately, you need to talk; you need to do it quickly; and, you need to do it easily. It's imperative that the design enables you, not hinders you. If we expose the cryptography to you, we're creating a barrier between the app and what you really want to do. Although you need what cryptography provides, it can't get in the way of you doing your job.

What We Need

We don't need better encryption; we need a better experience. As renowned experience designer Aral Balkan captured in his talk for Thinking Digital 2013, "Superheroes and Villains in Design": as users, we should approach design naively and let it tell us how it wants to be used. When we do this, we recognize the product for what it is, the expert; we should be able to trust it to make the right decisions and give us the affordances we expect. In the case of the journalist above, this implies several things about the user experience. Everything matters. You'll need to consider the right background and foreground colors, and typefaces as well, to prevent eye fatigue from straining to see what's being displayed. Also, you'll have to think about the average size of fingertips so as to prevent misfires; seconds lost to poor interaction can be costly. Oh, and the arrangement of objects on the display is a big deal, too; an object's function should be obvious. And then there's the fact that this journalist is likely to be in vastly different cultures. With that in mind, the symbols and colors you use must make sense within the context of the culture with which the source identifies.

The design should anticipate the needs of its users; the experience should fulfill their wants. The journalist doesn't want to encrypt and authenticate the data channel between himself and the source; the journalist wants to safely talk to his source. He needs the former, but wants the latter. Balkan's forthcoming project, Codename Prometheus, is focused on experience design in the consumer space, with a strong emphasis on protecting security, privacy, and human rights. This is a big step in the right direction of cultivating the experience for the consumer and solving the conceptual problems they care about (e.g., how can I communicate conveniently, but safely?), without burdening them with our own problems regarding the details (e.g., how can I make this app encrypt and authenticate communications?).

What all of this is trying to tell us is that we've been taking a monolithic approach to development for far too long. It's simply not enough for cryptographers to help developers properly implement; that's only one-half of real-world cryptographic design. What we absolutely must have are experience designers helping developers properly interface. Ignoring this carries on the tired, hapless campaign of "cryptography for the masses," which didn't materialize into the cypherpunk dream; by inviting it, however, we have a fair shot at helping those masses benefit from cryptography. In the cryptographer-to-developer relationship, cryptographers have the ability to work with developers on this problem; in the developer-to-consumer relationship, the consumer hasn't the expertise to work with the developer. Experience designers do, however; they speak to the needs and wants of the consumer

on their behalf. In other words, developers have a chance at getting the implementation right with cryptographers around; without experience designers around, however, there's little hope of them getting the interface right.

A Fish Called Mackerel

Mackerel is a cryptographic design paradigm that posits that practical cryptography is essentially a subset of product design. And because it's about products, it's about people, and the need for a holistic product design process that respects the roles of the people involved—cryptographers, developers, and consumers—by only asking them to make decisions that lie within their respective areas of understanding, and of which they understand the consequences. Ultimately, by focusing on the cryptographer-to-developer and developer-to-consumer relationships, the outcome will render the assurance of the underlying implementation, as well as the accessibility of the user interface, resulting in a product that's useful, by offering both utility and usability to the consumer, and that behaves securely and privately. In short, Mackerel is a developer-centric, consumer-targeted "conception-to-cellophane" approach to building a cryptographically enhanced product from the ground up; the goal is to optimize the GUI (interface accessibility) and API (implementation assurance), by looking at tried-and-true elements from both product design and security engineering.

The Mackerel framework is intended to operate similarly to a software development framework, where the design and development of a cryptographic product is modeled as a dissection of individual components that, although they all affect the overall goal of security and privacy, often require distinct approaches. For example, within this framework would be cryptographic threat modeling, where the intended application of a product and its operating environment are considered in order to determine applicable attacks and the appropriate cryptographic measures for mitigating them. This is clearly a security and privacy problem with a security and privacy answer; however, as the framework shifts from low-level to high-level, where you're dealing with usability factors and the overall experience of the product, you're dealing with a problem that can't be answered by security and privacy experts. (If we try to do so, we risk PGP 2.0: hard to break, but hard to use.) It can be answered by usability experts and those who design experiences for a living, which is what has been missing in the modern day process. Although a bad interface and experience can lead to a poor security and privacy decision, this doesn't mean the interface and experience are security and privacy problems or can be solved as such; it means that we can't solve the interface and experience problems without experts in those areas working alongside security and privacy experts. We currently having nothing of the sort, let alone a framework that involves both.

Once you birth cryptography into the real world, it becomes a small component in a large composite that has more non-cryptographic parts than cryptographic ones; having said that, you can't build a good cryptographic product if you involve cryptographers but not product designers. You certainly can't build a good cryptographic product if you think it's entirely a cryptographic problem, or even entirely a security and privacy problem. Mackerel models every core aspect of cryptography's evolution as a product, such that optimal decisions can be made, given the state-of-the-art know-how in cryptographic design, software development, and user experience design.

Lastly, let's tell you why Mackerel is called "Mackerel." At first glance, it might seem like just another entry into cryptography's long list of systems named after fish. Well, that's partially true, but there's a bit more. Integrity is as important a goal as confidentiality, if not sometimes more. After all, breaking confidentiality is the ability to passively eavesdrop, whereas breaking integrity is the ability to actively manipulate. You can imagine how the latter can render far worse results than the former, and even result in the loss of both. So, although encryption is supposed to handle confidentiality, it often can't, without authentication, and the standard way to go about that is through the use of a MAC, or message authentication code. If there's anything out of all of this research that we hope you learn, from a cryptographic point of view, it's that you should always use a MAC, or an AEAD mode that does both encryption and authentication, or die trying.

In order to pay homage to the glorious yet underappreciated MAC, it was befitting to choose as a moniker the fish whose name begins with "mac": the mackerel.

Acknowledgments

We immensely thank the vast number of people whose eyes and ears have been so graciously loaned over the past five years since our work on "green cryptography" first emerged. A distinguished thanks to Brian Snow for supporting our vision and for helping to shape it with his uncanny know-how and grasp of the level of assurance and accessibility we should expect from a cryptographic product.

References

[1] J. Troutman and V. Rijmen, "Green Cryptography: Cleaner Engineering through Recycling," *IEEE Security and Privacy*, vol. 7 (2009), pp. 71-73.

[2] J. Troutman and V. Rijmen, "Green Cryptography: Cleaner Engineering through Recycling, Part 2," *IEEE Security and Privacy*, vol. 7 (2009), pp. 64-65.

[3] J. Troutman, "Green Cryptography": extended drafts can be found at http://justintroutman.com, 2013.

[4] A. Dey and S. Weis, "Keyczar: A Cryptographic Toolkit," 2008.

[5] D. J. Bernstein, T. Lange, and P. Schwabe, "The Security Impact of a New Cryptographic Library," Cryptology ePrint Archive, Report 2011/646, 2011: http://eprint.iacr.org.

[6] D. J. Bernstein, "The Salsa20 Family of Stream Ciphers," in *New Stream Cipher Designs* (Springer-Verlag Berlin, 2008), pp. 84-97.

[7] D. J. Bernstein, "The Poly1305-AES Message-Authentication Code," in *Fast Software Encryption* (2005), pp. 32-49.

[8] A. Whitten and J. D. Tygar, "Why Johnny Can't Encrypt," Proceedings of the 8th USENIX Security Symposium, 1999.

[9] M. E. Zurko and A. S. Patrick, "Panel: Usable Cryptography: Manifest Destiny or Oxymoron?," in *Financial Cryptography* (2008), pp. 302-306.