ECRYPT CSA

H2020-ICT-2014 – Project 645421

ECRYPT – CSA

ECRYPT – Coordination & Support Action

D1.1
Challenges in Authenticated Encryption

Due date of deliverable: May 2015 (workshop) + September 2016 (white paper)
Actual submission date: 17. July 2015 (workshop) + 1. March 2017 (white paper)

Start date of project: 1 March 2015 Duration: 3 years

Lead contractor: Technische Universiteit Eindhoven (TUE) Revision 1.05

Change log

Version	Contents
1.05	improvements after discussions at DIAC 2016 and online
1.0	first release

Project co-funded by the European Commission within the H2020 Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission services)	
RE	Restricted to a group specified by the consortium (including the Commission services)	
CO	Confidential, only for members of the consortium (including the Commission services)	

Challenges in Authenticated Encryption

Editor
Daniel J. Bernstein

Contributors (alphabetical order; affiliations included for identification only)
Jean-Philippe Aumasson (Kudelski Security, Switzerland)
Steve Babbage (Vodafone, UK)
Daniel J. Bernstein (University of Illinois at Chicago, USA;
Technische Universiteit Eindhoven, Netherlands)
Carlos Cid (Royal Holloway, University of London, UK)
Joan Daemen (STMicroelectronics, Belgium;
Radboud Universiteit, Netherlands)
Orr Dunkelman (University of Haifa, Israel)
Kris Gaj (George Mason University, USA)
Shay Gueron (University of Haifa, Israel; Intel, Israel)
Pascal Junod (HEIG-VD, Switzerland)
Adam Langley (Google, USA)
David McGrew (Cisco, USA)
Kenny Paterson (Royal Holloway, University of London, UK)
Bart Preneel (KU Leuven, Belgium)
Christian Rechberger (Danmarks Tekniske Universitet, Denmark)
Vincent Rijmen (KU Leuven, Belgium)
Matt Robshaw (Impinj, USA)
Palash Sarkar (Indian Statistical Institute, Kolkata, India)
Patrick Schaumont (Virginia Tech, USA)
Adi Shamir (Weizmann Institute, Israel)
Ingrid Verbauwhede (KU Leuven, Belgium)

17. July 2015 (workshop) + 1. March 2017 (white paper)

Revision 1.05

The work described in this report has in part been supported by the Commission of the European Communities through the H2020-ICT program under contract H2020-ICT-2014 no. 645421. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Contents

Executive summary

Authenticated encryption is the cryptographer's front-line defense against attackers. It is the protective shield applied to every network packet. It is the foundation of security for medical devices, connected vehicles, the financial sector, the smart grid, and the Internet of Things. But is this shield actually being used? Is it actually working? Is it doing what the users actually need? Are industry practitioners listening to researchers? Are researchers listening to industry practitioners?

This white paper identifies critical ongoing problems whose solutions will need concerted community effort stretching years into the future. The challenges described in this white paper are classified into four categories:

- Chapter 1: The security target is wrong.
- Chapter 2: The interface is wrong.
- Chapter 3: The performance target is wrong.
- Chapter 4: Mistakes and malice.

This white paper does not mean to suggest that authenticated ciphers are *always* aiming at the wrong target. It is important to understand that, for many environments today, using an existing standard such as AES-128-GCM [19] is simple, safe, and efficient. However, it is equally important to understand that the existing standards *fail* to meet the needs of many other environments. The AES cipher [10] and the AES-GCM authenticated cipher are used as examples throughout this document to illustrate what can and does go wrong.

Audience

This white paper is aimed at several target groups:

- Practitioners and users, so that they can understand the most important limitations of today's cryptography.
- Researchers, so that they can help develop solutions to these problems.
- Managers, so that they can invest their cryptographic resources wisely.
- Standardization bodies, so that they can make better plans for future standards and updates in current standards.

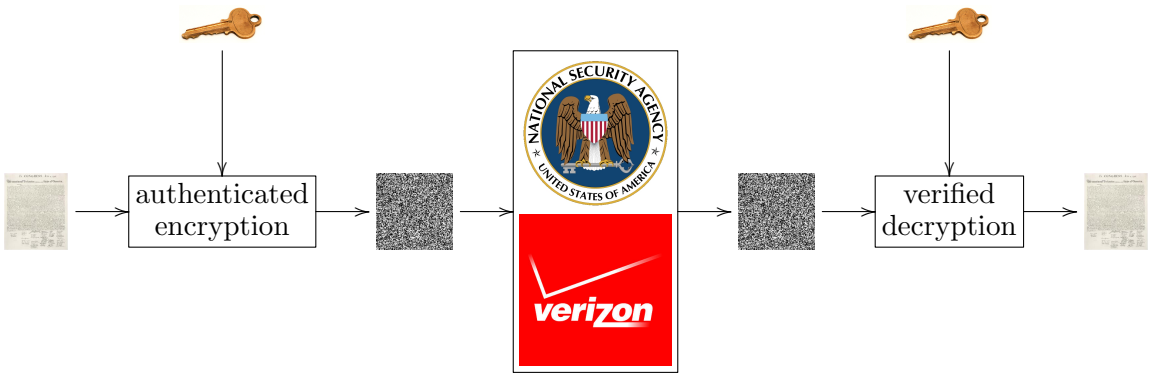
Framework and history

This white paper is produced by ECRYPT-CSA, a Coordination and Support Action sponsored by the European Union's Horizon 2020 programme. ECRYPT-CSA identified key players from around the world and solicited their input, receiving a tremendous range of feedback; 30 of these key players joined a private mailing list dedicated to this white paper. ECRYPT-CSA held a public 1-day workshop on 17 July 2015 in Utrecht, the Netherlands, attracting 10 of the key players, additional experts, and other interested parties. The workshop organized the full group of 36 attendees around a circle and featured a full day of intense discussions, with smaller group discussions over coffee and lunch in the same room. ECRYPT-CSA also kicked off a public web site `chae.cr.yp.to` at the end of June 2015 with an introductory page and a workshop page. The same web site will host future updates of this white paper.

Chapter 0

A brief introduction to authenticated encryption

Authenticated encryption allows two parties, say Alice and Bob, to exchange messages. The prerequisite is that Alice and Bob each have a copy of a secret key. Alice uses the secret key to **encrypt** a **plaintext** message, computing a scrambled **ciphertext**. Alice then sends the ciphertext through an untrusted communication channel:



Bob receives the ciphertext and uses the secret key to **decrypt** the ciphertext, computing the original plaintext. Subsequent messages from Alice to Bob (or from Bob to Alice) are encrypted and decrypted similarly.

0.1 Confidentiality

Authenticated encryption has two fundamental security goals. The first fundamental security goal is **confidentiality** despite espionage by the untrusted network. A spy, given the ciphertexts, should not be able to figure out anything about the plaintexts.

Of course, a spy who has a copy of the secret key can decrypt messages by simply performing the same computations that Bob performs. It is thus essential for Alice and Bob to maintain the secrecy of the secret key.

Confidentiality is limited by the exposure of **metadata**. The spy can see *who* is sending a ciphertext (namely Alice), and *when* the ciphertext is sent, and *how long* the ciphertext is

(which in turn reveals information about how long the plaintext is), and who is receiving the ciphertext (namely Bob).

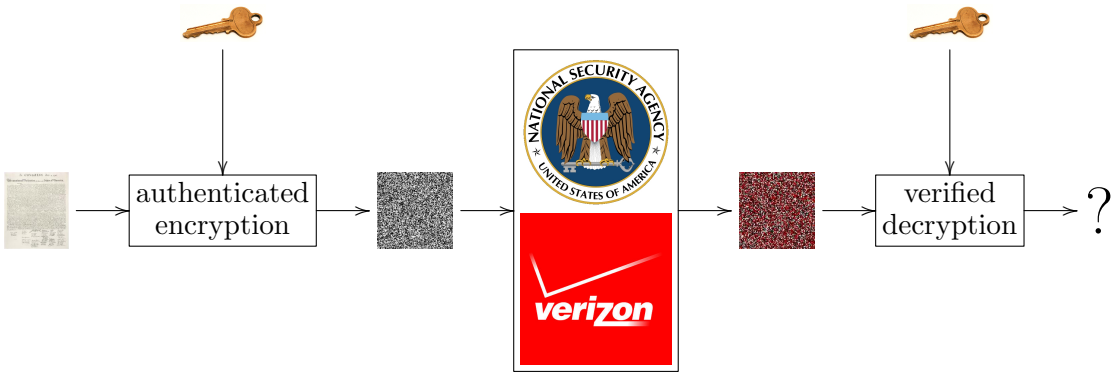
Confidentiality is also limited by the exposure of repeated plaintexts as repeated ciphertexts. For example, if Alice sends plaintext "WARM" and then plaintext "COLD" and then plaintext "WARM", the spy will see that the first and third ciphertexts are the same, and will deduce that the first and third plaintexts are the same, while the second plaintext is different.

One standard solution to the repeated-plaintext problem is for Alice and Bob to insert a unique message number (a **nonce**: "number used once") at the beginning of each plaintext, so that plaintexts never repeat: "1 WARM", "2 COLD", "3 WARM". Common practice is for this nonce to be an unencrypted counter, although this exposes metadata that might otherwise be more expensive to collect. A common variant is for the nonce to be chosen randomly (and long enough to avoid random collisions), although this raises questions about the quality and cost of the random-number generator. It is also possible to encrypt nonces.

Current standards and proposals for authenticated ciphers vary in how much damage is done if a message number repeats. At one extreme, AES-GCM loses all security in this scenario; the user is responsible for ensuring proper use of nonces. At the opposite extreme, some new proposals guarantee "maximum nonce misuse resistance": the only damage done by a repeated message number is exposure of whether the plaintext repeated. There are intermediate possibilities, and there is now considerable literature analyzing the resulting tradeoffs between security and performance.

0.2 Integrity

The second fundamental security goal of authenticated encryption is **integrity** despite sabotage by the untrusted network. Each ciphertext includes an **authenticator** that is verified as part of the decryption process, so any modification of the ciphertext by the untrusted network will be recognized:



Integrity does not guarantee **availability** of the correct plaintext: Alice and Bob need further mechanisms to react to a verification failure, somehow ensuring that the correct plaintext is sent again.

Integrity is limited by the ability of the network to **replay** messages. If Bob receives "PAY CHARLIE 1000 EUR" and then later "PAY CHARLIE 1000 EUR", was Alice in fact sending this message twice, or is the network maliciously copying the old message? A

standard solution is for the sender to include a unique message number at the beginning of each plaintext (which also helps confidentiality, as discussed above), and for the receiver to discard any message number used previously under the same key. If message numbers are always selected in increasing order then the receiver can simply remember the last message number accepted, and reject any message number that is not larger.

Integrity is also limited by the ability of the network to **reflect** messages. Reflection is a special type of replay: the network takes a message from Alice to Bob, and sends it as a message from Bob to Alice. A standard solution is for plaintexts to explicitly identify their direction. Another standard solution is to include the Alice-to-Bob or Bob-to-Alice metadata as **associated data** that is also authenticated without being encrypted.

0.3 Performance

Exponential increases in computer power have been accompanied by exponential increases in the amount of data transmitted. It is essential for authenticated ciphers to be able to keep up with all this data: otherwise users will be forced to expose some data to espionage and sabotage. Obtaining the best possible tradeoffs between security and performance, and in particular the best possible security subject to the users' performance constraints, has always been one of the central goals of research in this area.

Common platforms for cryptographic software include 8-bit, 16-bit, and 32-bit embedded CPUs; 32-bit and 64-bit smartphone CPUs; 32-bit and 64-bit laptop CPUs; 64-bit desktop CPUs; and 64-bit server CPUs. Cryptographic software performance is measured primarily by the number of CPU clock cycles required to process messages of various lengths. This number can vary between encryption, decryption of valid ciphertexts, and rejection of forged ciphertexts.

There are also many chips that include cryptographic hardware. Important performance metrics for hardware include throughput (how much data can be processed per second), latency (how much real time elapses before results are available), chip area (both the area used to store state and the area used for computation), power, and energy consumption. Hardware implementations can be prototyped on field-programmable gate arrays (FPGAs) before being burned into application-specific integrated circuits (ASICs); in many cases FPGAs are already fast enough to be used in production, and their agility often reduces total costs.

Chapter 1

The security target is wrong

Often an authenticated cipher is designed to achieve security against one type of attack, while the user actually needs security against another type of attack. In some cases the cipher ends up being broken because the security targets were set too low; see, e.g., the side-channel example below. In other cases the cipher ends up being unnecessarily expensive, hampering deployment in performance-sensitive environments, because the security targets were set too high; see, e.g., the 80-bit example below.

1.1 Side-channel attacks—the security target is too low

The standardization process for the AES cipher carefully considered ways that an attacker might learn something from inspecting the AES outputs. The process concluded that AES was strong. However, a decade ago Osvik, Shamir, and Tromer [22] presented a 65-millisecond attack successfully stealing AES keys used for disk encryption. This attack did not inspect the AES *outputs*; instead it made observations of *metadata* regarding the AES computation, specifically the time required to access cache lines evicted by the AES computation.

Side-channel attacks are applicable far beyond AES, and today remain one of the top threats against real-world authenticated encryption, as illustrated by various TLS implementations being broken by the "Lucky Thirteen" [2] and "Lucky Microseconds" [1] timing attacks. It is well understood in theory how to avoid all leakage of secrets through timing, but this has only limited support in typical cryptographic software libraries. One of the underlying issues is that constant-time software creates performance problems in most ciphers and modes of operation; this in turn reflects a lack of attention to side-channel attacks at the cryptographic-design stage.

Even worse, attackers who are physically close to the device under attack (or who have control over enough nearby sensors, such as smartphone sensors) can collect information from many more side channels: power consumption, electromagnetic radiation, etc. It is not at all clear how to protect secrets from leaking through these side channels. The security of proposed "countermeasures" is poorly understood, and these countermeasures again create performance problems in most ciphers.

1.2 Birthday attacks—the security target is too low

There is a proof guaranteeing that the standard AES-GCM authenticated cipher is *almost* as secure as the underlying AES block cipher. Specifically, it is *almost* true that an attack against AES-GCM cannot be more successful than an attack against AES. But it is not exactly true. To be precise, an attack using q blocks of AES-GCM ciphertext can be more successful than an attack against AES; what the proof guarantees is a particular limit on the extra success chance. The limit is roughly $q^2/2^{123}$; the exact formula does not matter for the following comments.

If $q = 2^{62}$ then the AES-GCM security guarantee is useless. Even if q is just 2^{50}, the extra success chance could be as high as one in eight million, which might sound small but is not a success chance that cryptographers would tolerate for high-value data. Similar comments apply to most authenticated ciphers, and in many cases researchers have developed **birthday attacks** showing that there really is a gap between the security of AES and the security of systems built on top of AES.

The very recent "SWEET32" attack [5] is a low-cost birthday attack against 64-bit block ciphers allowed in TLS and OpenVPN. SWEET32 reaches a high success chance using $q \approx 2^{37}$. AES has a larger block size, so AES-GCM is not broken at such low cost, but from this perspective the security margin of AES-GCM is surprisingly small.

There has been some initial work on **beyond-birthday-bound** cryptography, which obtains meaningful AES-based guarantees for considerably larger values of q by using AES in a more complicated way, and on **bigger-birthday-bound** cryptography, which obtains meaningful guarantees by replacing AES with "larger-block" ciphers. See, e.g., [13] for a beyond-birthday-bound authenticated cipher and [9] for a bigger-birthday-bound authenticated cipher. The challenge is to understand the fundamental cost of security as a function of the amount of data transmitted.

1.3 Data limits—the security target is too high

A different response to "birthday attacks" is to design security systems that enforce controls over how often a secret key is used. These controls usually require some key-switching complexity elsewhere in the system, but almost all systems have this complexity anyway.

As a concrete example: NIST's GCM standard says that, when its 96-bit nonces are generated randomly, the "total number of invocations of the authenticated encryption function shall not exceed 2^{32}, including all IV lengths and all instances of the authenticated encryption function with the given key". It is not unusual for an application to send more than 2^{32} messages, and then this requirement forces the application to switch keys. This requirement is important: an application that actually sends 2^{40} messages under the same key would have an unacceptably high chance of nonces colliding, more than one chance in a million, and would then lose all security.

The conventional view is that an attack using 2^{50} ciphertexts and 2^{100} computations would constitute a "break" of a cipher that has a 128-bit key. However, attackers are constrained not only by the computation that they can afford, but also by limits on the amount of data provided to them (known plaintexts, chosen plaintexts, chosen ciphertexts). A consequence of limiting keys to, e.g., 2^{20} ciphertexts is that users are protected against attacks that use 2^{50}, 2^{40}, or even 2^{30} ciphertexts. The challenge here is for the cipher designer to build more

efficient ciphers by taking advantage of this protection.

This research direction is controversial. One argument against checking the nominal number of ciphertexts used by an attack is that attacks will often work with fewer ciphertexts at a moderate (linear or quadratic) loss in success probability; this must be carefully taken into account. Another argument against building ciphers in this way is that the security of the cipher now depends on other parts of the system to switch keys quickly enough. One counterargument is that the key-switching part of the system is also security-critical and needs careful cryptographic review in any case; this review can also check whether keys are switched as often as the cipher requires.

1.4 Attack economics—the security target is too high

An even more controversial research direction is to assess how much the attacker *gains* from an attack, compare it to the cost of the attack, and choose key sizes so that the cost outweighs the benefit.

For example, many **lightweight** ciphers use 80-bit keys, even though it is clearly feasible for a well-funded attacker to search through all 2^{80} possible keys. The usual argument that 80-bit keys are sufficient is that the attacker's expected benefit is smaller than the attacker's expected cost, so the attacker will not carry out the attack.

The usual counterargument is that it is actually quite tricky to evaluate the costs and benefits of an attack. The challenge is to do this correctly. One difficulty is that many complicated layers of security systems are built under the assumption that the underlying crypto is unbroken; evaluating the impact of key recovery, plaintext recovery, forgery, etc. requires a holistic understanding of the entire system. Another difficulty is that, if breaking *one* key is feasible, then breaking *many* keys often has surprisingly little extra cost; see, e.g., [8] and [4].

Similar comments apply to short authenticators, say 32-bit or 16-bit authenticators. The Internet of Things will have very large numbers of devices with very small amounts of data to send each time they communicate, and the overhead for transmitting a long authenticator can be quite problematic, especially for devices that run off batteries. It is of course unacceptable if an attacker can steal some money or fire a missile by forging a single 32-bit authenticator; but often authenticators are protecting low-value data, and often preexisting redundancies in data effectively increase the length of an authenticator. The challenge is again to accurately evaluate the costs and benefits of an attack.

Perhaps performance requirements prevent deployment of 128-bit keys and 128-bit authenticators. Perhaps the most that the user can afford is an 80-bit key and a 32-bit authenticator. The only hope for security is then to analyze all of these complications and redesign systems accordingly.

1.5 Quantum computers—the security target is too low

Michele Mosca, Deputy Director of the Institute for Quantum Computing at the University of Waterloo, has estimated a "1/7 chance" of a useful quantum computer—one that can break RSA-2048—by the year 2026 and a "1/2 chance" by 2031. See [21].

Quantum computation would not have such dramatic effects on authenticated ciphers (Shor's algorithm [23], the quantum algorithm that breaks RSA-2048, is not applicable to most ciphers and authenticators), but it would have *some* effect. Specifically, the same

computers would also be able to run Grover's algorithm [11], which searches through 2^k possible keys using only $2^{k/2}$ quantum operations.

One way to react to Grover's algorithm is to replace k-bit keys with $2k$-bit keys: for example, switch from AES-128 to AES-256. However, this is likely to be overkill, for at least three reasons. First, the number of rounds in AES-256 was selected to provide 2^{256} security against more advanced attacks than brute-force search; it seems likely that quantum computation would have less impact upon these attacks than upon brute-force search, so fewer rounds are required. Second, Grover's algorithm is inherently serial, so the speedup that it provides is limited by the time available, not by the product of time and hardware area. Third, even under optimistic projections regarding progress in quantum computation, the ultimate cost of qubit operations is likely to be considerably more expensive than the ultimate cost of bit operations.

The challenge here is to analyze what is actually required for security against future quantum computers. Note that attackers storing ciphertext today can target the ciphertext using quantum computers in the future, and upgrading users to new cryptographic systems takes time (especially in applications where the old systems are built into long-lasting hardware units that are hard to upgrade, such as typical smartcards and implanted medical devices), so there is an urgent need to respond to the threat of quantum computers many years before quantum computers are built.

Chapter 2

The interface is wrong

Encryption has slowly gained in sophistication through a series of focused competitions over the past two decades:

- The AES competition called for **block ciphers**. A block cipher encrypts fixed-length blocks: in particular, AES encrypts 16-byte blocks.
- The eSTREAM competition called for **stream ciphers**. A stream cipher is more flexible than a block cipher: it encrypts *variable-length* messages.
- The ongoing CAESAR competition called for **authenticated ciphers**. An authenticated cipher encrypts *and authenticates* variable-length messages.

One can build a stream cipher and an authenticator from a block cipher: for example, the AES-CTR stream cipher and the AES-OMAC authenticator [14] use AES. One can build an authenticated cipher from a stream cipher and an authenticator: for example, the AES-EAX authenticated cipher [3] uses AES-CTR for encryption and AES-OMAC for authentication. This might sound simple, but there are many options for the details at each layer, with many opportunities to damage security and performance. For example, AES-EAX is much slower than AES-GCM; "EAXPrime", an ANSI standard that tried to streamline EAX, was shown in 2013 [20] to be easily breakable.

A cipher designer who directly targets an authenticated cipher produces simpler, smaller, faster, more robust designs than a cipher designer who merely targets a block cipher. Some of the ciphers selected for the third round of the CAESAR competition use different internal layering, for example building authenticated encryption out of a **tweakable block cipher** or a **permutation**; some of the ciphers have just one layer; all of the ciphers have clear advantages over AES-GCM.

To summarize, the shift of focus from block ciphers up to stream ciphers and then to authenticated ciphers is obviously progress. However, there are still some important gaps between what today's authenticated ciphers provide and what applications actually need.

2.1 Streams

What network applications typically want is authentication and encryption for a stream of data from a sender to a receiver, or for streams of data both ways between a client and a server.

An authenticated cipher might at first sound like a perfect fit for the job of protecting (e.g.) a web page sent from a web server to a web browser. The server simply formats the web page as a message, encrypts and authenticates the message using the authenticated cipher, and sends the ciphertext to the browser through the Internet's Transmission Control Protocol (TCP). The browser verifies and decrypts the message and displays the resulting web page.

However, this authentication structure means that a forgery cannot be discovered until the entire web page is transmitted. The web browser actually starts displaying parts of the web page as soon as those parts are received; this is also what the user wants to see. This causes two levels of serious problems:

- Many ciphers are vulnerable to **release of unverified plaintext**. This means that, if a cipher implementation starts decrypting a forged message and sending out the results, and if those results are visible somehow to the attacker, then the attacker can forge or decrypt *other* messages. The ciphers are not designed to be secure in this scenario.
- Even when ciphers are safe against release of unverified plaintext, the attacker is free to modify almost all of the data being given to the browser. There is no guarantee that the forgery will *ever* be detected: the attacker might flood the network just before the authenticator is sent.

The general picture is that the receiver of a stream often acts immediately upon the data it has received, so this data needs to be immediately (**incrementally**) verified.

A closer look shows even more problems. TCP splits streams of data between a client and a server into limited-length **packets** sent through the network; for example, a web page sent by a web server usually fills thousands of packets, while the initial request for the web page might have fit into a single packet. Sometimes packets are randomly lost (most commonly because of unreliable radio links or temporary network congestion); TCP includes mechanisms to acknowledge packets that were received and retransmit packets that were lost. Unfortunately, this *metadata* is subject to forgery even if the *contents* of every packet are authenticated. An attacker can destroy a TCP connection by sending a single forged packet; this denial-of-service attack is much less expensive than flooding the network.

The problems described above are concerned solely with integrity and availability, protecting against forgeries. Even more problems arise when the user needs to protect the confidentiality of metadata (for example, how much data is being sent at which moments) or to provide "forward secrecy", protecting confidentiality against future theft of secret keys. The overall challenge here is to integrate authenticated encryption into networking in a way that provides the security properties that applications actually need for network data.

2.2 Files

Further problems appear in applications that authenticate and encrypt access to *files*: for example, files stored on a disk that might be stolen, or files stored on an untrustworthy cloud provider. One can again think of a file as a message to be encrypted and authenticated all at once, but again the reality is that users expect to be able to process *parts* of a file without first buffering and verifying the *entire* file. Sometimes a smartphone is accessing a huge file stored on the cloud and does not even have enough storage to hold the entire file, even when latency is not an issue. An analogous problem appears when a small Trusted Execution Environment is accessing a much larger amount of data stored in untrusted RAM.

Deployed disk encryption today is typically "AES-XTS", which is vulnerable to active attacks. The security picture is generally getting worse rather than better, as illustrated by Microsoft significantly weakening its BitLocker disk-encryption system starting in Windows 8 for the sake of "performance on low-powered devices". See [17].

The extra challenge for files, compared to streams, is that portions of a file can be accessed *in any order*. A network delivers packets but does not try to store packets; a disk or cloud provider has the extra feature of storing large amounts of data, giving the receiver the opportunity to access the data repeatedly, to search through different positions in the data, etc.

2.3 Noisy channels

Physical communication channels, such as the radio waves used by WiFi and mobile telephones, always have some amount of noise. Some bits are flipped from 0 to 1 or 1 to 0; sometimes these errors occur in bursts; sometimes stretches of data are erased.

Standard practice is to apply an **error-correcting code** to each message, mathematically expanding the message in a way that allows the receiver to correct a few errors and often to detect more errors. However, if the message is already an authenticated ciphertext, then the original plaintext was already mathematically expanded in a way that allows the receiver to reliably detect any number of errors. Surely this two-stage combination is not the most efficient way to build what the user actually needs, namely error-correcting authenticated encryption, allowing the receiver to correct a few errors and reliably detect any number of errors.

Part of the challenge here is mathematical: optimizing the design of error-correcting authenticated ciphers, and in particular error-correcting message-authentication codes. Another part of the challenge is network engineering: today's networks are typically built as layers, with error correction handled at a different layer from cryptography, so deploying an error-correcting authenticated cipher would require the layers to be rearchitected.

2.4 Software engineering and hardware engineering

Cryptographic software libraries provide the interfaces used by practically all cryptographic software. Often there is a severe mismatch between these interfaces and the abstract interfaces defined in cryptographic papers.

For example, most libraries provide "streaming" interfaces to encryption and decryption, allowing a few bytes of data to be encrypted or decrypted at a time, even though most papers design and analyze ciphers for a simpler all-at-once interface. The software interface makes it easy for applications to release unverified plaintext, creating the problems described in Section 2.1. As another example, items labeled as "nonce" or "random" in library interfaces usually have many caveats that do not appear in papers.

Similar comments apply to cryptographic hardware. The challenge for both software and hardware is to eliminate the gap between the interfaces analyzed in papers and the interfaces actually used in the real world.

Chapter 3

The performance target is wrong

Modern authenticated encryption has low enough cost for many applications, but not all. Performance concerns often push security backwards. The problematic cases usually feature a separation between the type of cost that the cipher designer optimized for and the type of cost that actually matters for the user.

3.1 Denial-of-service attacks

Authenticated ciphers are usually optimized for the cost of handling *legitimate* data: the costs of encrypting and authenticating plaintext, and the costs of verifying and decrypting a *legitimate* ciphertext. This focus on legitimate traffic fails to acknowledge the reality faced by large Internet providers: these providers overprovision their servers so that they can handle distributed denial-of-service attacks.

Sometimes the attacker has enough resources to flood the network routers, and then some legitimate data is prevented from getting through no matter how ciphers are chosen. Sometimes, however, the network is adequately provisioned but a denial-of-service attack targets the *computation* on CPUs. The challenge here is to minimize the cost of rejecting forged packets.

3.2 Very short inputs

Another increasingly common scenario is that an authenticated cipher is applied to many *small* messages. Authenticated ciphers are normally optimized for the cost of handling *long* messages: for example, there has been very little attention to the costs of internally handling messages using large-block permutations, even though large blocks force short messages to be padded to the next block boundary. The challenge here is to minimize overhead.

3.3 Higher-level protocols

Authentication and encryption are best known as mechanisms to directly protect user data, but they are also applied inside fancier cryptographic protocols, and their costs inside these protocols are often quite different from their direct costs. Very recent work has begun to explore the possibility of designing new ciphers as tools for multiparty computation and fully homomorphic encryption.

3.4 Flexibility

Google and CloudFlare have both announced that they are now using the "ChaCha20" cipher (and the "Poly1305" authenticator) for a significant fraction of their TLS traffic. The reason is that ChaCha20 is faster than AES at encrypting data on typical 32-bit smartphones (and similarly Poly1305 is faster than the authenticator in GCM). See [6], [24], and [16].

On the other hand, typical 64-bit smartphones, desktop CPUs, and server CPUs have fast AES instructions (and GMAC-related instructions) in hardware, making AES (and GMAC) faster than ChaCha20 (and Poly1305) on those CPUs. From the perspective of a typical server at Google or CloudFlare, if the client also has AES hardware then using AES obviously saves time for both the client and the server. The tricky case is that the client does *not* have AES hardware: then choosing AES saves time for the server, while choosing ChaCha20 saves time for the client. Google and CloudFlare have chosen the second option, in effect paying performance penalties on their servers to save time for smartphones.

Analogous problems occur when information is communicated between, e.g., a swarm of tiny Internet of Things devices and a cloud of busy Internet servers. These problems are quantitatively even more severe than the smartphone/server problems, since performance on tiny devices is quite poorly correlated with performance on servers. Huge gaps are caused by variations in hardware capabilities, and also by variations in operational context, as illustrated by a server processing many messages at once.

Is there a way to resolve this type of tension between cipher performance on different platforms? The challenge here is to build *one* cipher that meets the users' performance requirements across a broad range of current and future platforms, taking account of what is actually available in hardware.

3.5 CPU evolution

Another challenge is to jointly optimize the design of authenticated ciphers and the design of CPUs. This is not the same challenge as designing authenticated ciphers for hardware: it is heavily constrained by the fact that CPU designers have to optimize their CPUs to be good at many different tasks simultaneously, not just cryptography. CPU hardware area devoted to one task is area taken away from other tasks. An instruction useful for many tasks is more attractive than an instruction useful for only one task. Furthermore, CPUs are expected to preserve compatibility—once an instruction is added to the CPU's instruction list, it stays in the list—so CPU designers tend to be quite conservative, adding instructions only if they are convinced that the instructions will have long-term value.

Chapter 4

Mistakes and malice

Attackers search for security failures inside the systems used to authenticate and encrypt data. It is of course essential for the cryptologic community to carry out comprehensive public **cryptanalysis**, finding and eliminating all weaknesses before systems are deployed. But this is much easier to say than to do. Cryptanalysis has a vast scope, including analysis of **primitives** such as AES, **modes of operation** such as GCM, and **implementations** in software and hardware. The state of the art in research is constantly advancing and is far beyond what has been automated. There are many opportunities to *accidentally* create security flaws that slip past review, especially at the complicated interfaces between primitives, modes of operation, and implementations. There are also many opportunities to *maliciously* insert security flaws into systems.

4.1 Error-prone designs

Cryptographers and cryptanalysts generally assume that designs are implemented and used exactly as specified. The cryptographer designs systems under this assumption, trying to protect the user against any subsequent actions by a malicious attacker. The cryptanalyst evaluates systems under this assumption, trying to figure out whether the user is protected. In reality, however, implementations often deviate from what was specified, and these deviations are often a huge source of security holes.

It is traditional for designers to blame implementors for any failures that the implementors could possibly have avoided. However, in many cases implementors can justifiably blame designers for creating systems that are hard to implement correctly, hard for the implementor to test, and hard for anyone else to test. Consider, for example, Gueron and Krasnov [12] reporting a serious security hole in AES-GCM software that had passed tests and that was ready for deployment in the next release of the OpenSSL cryptographic library.

It is easy to say that simplicity helps avoid errors, but it is hard to pinpoint what exactly this means. The challenge here is to identify detailed design principles for implementor-friendly authenticated encryption.

4.2 Unverifiability

The AES-GCM standard explains how to authenticate and encrypt a message. This explanation is designed solely for human comprehension; it is not in a formal language understood

by computers.

There are many software libraries in various programming languages that claim to implement AES-GCM. Auditing these claims is an extremely tedious project, involving extensive human effort to read, verify, and compare every line of code. Perhaps some of these libraries have bugs, as illustrated by the Gueron–Krasnov discovery mentioned above. The hardware situation is even more difficult: there are very few cryptographers who are able to confidently compare a hardware description written in Verilog to a reference software implementation written in C.

There are some verification tools that in limited cases allow computers to verify that two programs (in source code or object code) compute the same function. The challenge here is to close the gaps between the capabilities of these tools, the high-level structure needed by designers, and the low-level optimizations needed by implementors. Particularly challenging would be to build a domain-specific language powerful enough to be simultaneously used by (1) designers specifying authenticated ciphers, (2) implementors formally verifying that their implementations (including side-channel-protected implementations and fault-tolerant implementations) match these specifications, and (3) evaluators investigating cipher security.

4.3 Miscommunication of security prerequisites

As noted above, there is a proof guaranteeing that AES-GCM is almost as secure as AES. This illustrates the following standard pattern: a mode of operation M (for example, GCM) is accompanied by a theorem stating that *if* the underlying primitive P (for example, AES) is secure then the cipher P-M is secure.

Unfortunately, a closer look shows that the assumption that P is "secure" is actually a cryptic list of highly technical assumptions about P, and similarly the conclusion that P-M is "secure" makes assumptions about user behavior, with many qualitative and quantitative traps and pitfalls. These lists are tedious to read even for cryptographic experts. One cannot simply ignore the details: there are many "secure" choices of P and M for which P-M turns out to be breakable because of a mismatch between the detailed notions of "security".

For example, consider again the successful SWEET32 attack against TLS (see Section 1.2). This attack targets CBC-mode encryption, which has "SemCPA advantage" guaranteed to be at most the "PRP advantage" of the underlying block cipher plus $q^2/2^n$, where q is the number of blocks encrypted by an attack and n is the number of bits in each block. A block cipher is typically declared to be "secure" if the "PRP advantage" is small, but this fails to guarantee a small "SemCPA advantage" for CBC. The formula $q^2/2^n$ implicitly puts another security requirement upon the cipher, a quantitative requirement for n to be large enough, depending on q; SWEET32 exploits the fact that ciphers often fail this requirement. ISO claims that CBC "will be safe" as long as rekeying takes place every "$2^{n/2}$ blocks", but this claim is flatly wrong: the $q^2/2^n$ formula provides no security assurance if $q = 2^{n/2}$, and SWEET32 successfully breaks security in this scenario.

The point of this section is not birthday attacks in particular. The point of this section is the severe communication failure, a much broader issue. Designers blame implementors for not thinking through the $q^2/2^n$; not managing keys properly; not managing nonces properly; not managing randomness properly; releasing unverified plaintexts; not protecting themselves against side-channel attacks; etc. Implementors blame designers for building an incomprehensible minefield. The challenge here is to build an infrastructure that is *successful* at

communicating security requirements among the people jointly building a security system.

4.4 Incorrect proofs

The GCM security proof [19] was published in 2004. Iwata, Ohashi, and Minematsu discovered in 2012 [15] that the security proof was wrong. With considerable effort they constructed a replacement proof, but with a quantitatively worse security guarantee.

The security proof [18] for XCBv2, a standard disk-encryption mode designed as an improvement to XTS (see Section 2.2), was published in 2007. Chakraborty, Hernandez-Jimenez, and Sarkar discovered in 2013 [7] that the security proof was wrong. They constructed a replacement proof for some message lengths, again with a quantitatively worse security guarantee. For other message lengths they presented an *efficient attack* breaking XCBv2.

Security proofs are among the most difficult parts of cryptography to audit, and often contain errors, as these examples illustrate. The challenge here is to build a new world of proofs that covers everything needed for authenticated encryption while confidently eliminating these errors.

4.5 Malicious cryptographic software and hardware

The Snowden revelations have drawn a new level of attention to ways that cryptography can be silently compromised by providers of cryptographic hardware and software through covert channels, kleptography, algorithm substitution, etc. For example, an attacker with some control over a random-number generator can leak secrets by controlling the choice of random nonces inside a randomized authenticated cipher.

Open-source software is obviously helpful, allowing any number of researchers around the world to check for security problems; but it is obviously not a guarantee of security. The challenge here is to design verification tools that protect not merely against *accidents* but also against *malice*. One can draw an analogy here to the ways that messages are encoded for network transmission in coding theory and in cryptography: coding theory is asked to protect merely against accidents, while cryptography is also asked to protect against malice.

Even worse, cryptographic software is compiled by an untrustworthy compiler and run on an untrustworthy laptop that contains untrustworthy chips. Even if the laptop manufacturer is honest and has a completely secure supply chain, perhaps the laptop was tampered with in transit, or tampered with while left unattended. It is particularly challenging to address the gap between verifying the security properties of software for authenticated encryption and verifying the security properties of the device actually relied upon by cryptographic users.

Bibliography

[1] Martin R. Albrecht and Kenneth G. Paterson. Lucky microseconds: A timing attack on Amazon's s2n implementation of TLS. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 622–643, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[2] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky Thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[3] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407, New Delhi, India, February 5–7, 2004. Springer, Heidelberg, Germany.

[4] Daniel J. Bernstein. Break a dozen secret keys, get a million more for free, 2015. https://blog.cr.yp.to/20151120-batchattacks.html.

[5] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN, 2016. https://sweet32.info.

[6] Elie Bursztein. Speeding up and strengthening HTTPS connections for Chrome on Android, 2014. https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html.

[7] Debrup Chakraborty, Vicente Hernandez-Jimenez, and Palash Sarkar. Another look at XCB. Cryptology ePrint Archive, Report 2013/823, 2013. http://eprint.iacr.org/2013/823.

[8] Sanjit Chatterjee, Alfred Menezes, and Palash Sarkar. Another look at tightness. In Ali Miri and Serge Vaudenay, editors, *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 293–319, Toronto, Ontario, Canada, August 11–12, 2012. Springer, Heidelberg, Germany.

[9] Simon Cogliani, Diana-Stefania Maimuţ, David Naccache, Rodrigo Portella do Canto, Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár. OMD: A compression function mode of operation for authenticated encryption. In Antoine Joux and Amr M. Youssef,

editors, *SAC 2014: 21st Annual International Workshop on Selected Areas in Cryptography*, volume 8781 of *Lecture Notes in Computer Science*, pages 112–128, Montreal, QC, Canada, August 14–15, 2014. Springer, Heidelberg, Germany.

[10] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES—The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[11] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th Annual ACM Symposium on Theory of Computing*, pages 212–219, Philadephia, PA, USA, May 22–24, 1996. ACM Press.

[12] Shay Gueron and Vlad Krasnov. The fragility of AES-GCM authentication algorithm. Cryptology ePrint Archive, Report 2013/157, 2013. http://eprint.iacr.org/2013/157.

[13] Tetsu Iwata. Authenticated encryption mode for beyond the birthday bound security. In Serge Vaudenay, editor, *AFRICACRYPT 08: 1st International Conference on Cryptology in Africa*, volume 5023 of *Lecture Notes in Computer Science*, pages 125–142, Casablanca, Morocco, June 11–14, 2008. Springer, Heidelberg, Germany.

[14] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-key CBC MAC. In Thomas Johansson, editor, *Fast Software Encryption – FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153, Lund, Sweden, February 24–26, 2003. Springer, Heidelberg, Germany.

[15] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 31–49, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[16] Vlad Krasnov. It takes two to ChaCha (Poly), 2016. https://blog.cloudflare.com/it-takes-two-to-chacha-poly/.

[17] Micah Lee. Microsoft gives details about its controversial disk encryption, 2015. https://theintercept.com/2015/06/04/microsoft-disk-encryption/.

[18] David A. McGrew and Scott R. Fluhrer. The security of the extended codebook (XCB) mode of operation. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *SAC 2007: 14th Annual International Workshop on Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 311–327, Ottawa, Canada, August 16–17, 2007. Springer, Heidelberg, Germany.

[19] David A. McGrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004: 5th International Conference in Cryptology in India*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355, Chennai, India, December 20–22, 2004. Springer, Heidelberg, Germany.

[20] Kazuhiko Minematsu, Stefan Lucks, Hiraku Morita, and Tetsu Iwata. Attacks and security proofs of EAX-prime. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 327–347, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany.

[21] Michele Mosca. Cybersecurity in an era with quantum computers: Will we be ready? Cryptology ePrint Archive, Report 2015/1075, 2015. http://eprint.iacr.org/2015/1075.

[22] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20, San Jose, CA, USA, February 13–17, 2006. Springer, Heidelberg, Germany.

[23] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, New Mexico, November 20–22, 1994. IEEE Computer Society Press.

[24] Nick Sullivan. Do the ChaCha: better mobile performance with cryptography, 2015. https://blog.cloudflare.com/do-the-chacha-better-mobile-performance-with-cryptography/.