What Every Quantum Researcher and Engineer Should Know about Classical Cryptography

Rodney Van Meter[1, a)]

Faculty of Environment and Information Studies, Keio University.

(Dated: 26 June 2020)

The potential impact of Shor's factoring algorithm on public key cryptography and the potential of quantum key distribution to replace key agreement mechanisms such as Diffie-Hellman are two of the first facts that quantum researchers learn, but many researchers remain vague on how those mechanisms fit into the overall goal of achieving confidentiality, integrity and availability in classical communications. We describe the relationship between cryptography and communication protocols, with emphasis on key lengths, rekeying, block sizes and known mathematical attacks on various ciphers. We primarily focus on Internet encryption, but also introduce WiFi and cellular telephone networks. This information will help quantum researchers understand when quantum computers will affect classical cryptographic practice and discover new avenues for applying quantum technology to the overall problem of secure communication.

PACS numbers: Valid PACS appear here
Keywords: Suggested keywords

CONTENTS

a)Electronic mail: rdv@sfc.wide.ad.jp.

I. INTRODUCTION

⇒Comments on the structure/outline are in blue.

⇒Comments on what to write and other notes to myself are in forest green.

⇒Document status: Second, still drafty draft. Nothing really on post-quantum crypto, not enough on bitcoin, not enough on quantum attacks in symmetric crypto. It'll have to do.

⇒At the moment, there are zero figures in this document, which absolutely must be remedied.

Those who study the fields of quantum computing and quantum communication quickly become familiar with Shor's algorithm for factoring large numbers and calculating discrete logarithms[1], and with quantum key distribution[2–9]. These two independent discoveries both impact classical cryptography, one by making it more vulnerable, the other by making it stronger – in theory.

However, the direct, organized study of cryptography often ends there, despite the importance of cryptography as a driving force in funding and potential deployment of quantum technology. The target audience for this is primarily quantum computing researchers who are familiar with Shor's algorithm, Grover's algorithm and QKD, since those are among the first things you learn, but who have only a very rough idea of what it means to actually encrypt data and to use encryption in a real-world setting. This document will help to orient such readers so that they can communicate effectively with the classical communications and cryptography communities, and help them to ask the right research questions, such as:

- What QKD key generation rate is appropriate for quantum-protected activities such as:
  - e-commerce, especially web browsing; and
  - intra-organization network-to-network connection?

  Especially, under what performance conditions will existing mechanisms be replaced effectively one-for-one, and under what conditions will new capabilities be enabled?
- When will quantum computers be able to effectively attack:
  - the generation of cryptographic keys used for communications;
  - bulk data encryption itself;
  - digital signatures and hashes; and
  - blockchain?
- What mathematical techniques commonly used in cryptography might be adaptable to use in new quantum-based attacks?
- What is the response of the classical cryptographic and Internet protocol and operations communities to the potential development of quantum computers?

Addressing all of these questions in depth in a single document is impossible, but to help quantum researchers and engineers effectively answer these questions in concrete ways that help guide their own work, this document introduces four key areas of modern cryptographic practice and associated historical background:

1. the mathematics of encryption, including how math can be used to attack encryption;
2. how encryption is incorporated into complete communication systems;
3. how those systems are used to achieve the "CIA" goals of confidentiality, integrity and availability; and
4. how cryptography influences and is influenced by policy, both corporate/organizational and governmental.

What this document is *not*:

- a survey or tutorial on either QKD or Shor's algorithm;
- a full survey of the massive amounts of clever research done in quantum security algorithms (e.g.,[10–13]);
- a survey of techniques for certifying the quantumness of states[14]; or
- a full tutorial on cryptography.

A. Encrypted Communications

An encrypted conversation involves, at the macro level, three phases:

1. Authentication: proving that you are who you say you are;
2. Key generation: creating the keys used for bulk data encryption, possibly including regeneration of keys mid-connection (*rekeying*); and
3. Encryption/sending/decryption of the user's valuable bulk data.

That's a bit of an oversimplification, as we'll see below when we talk about IPsec (Sec. IV A), but good enough for now.

Rekeying, the changing of the cryptographic keys used during long communication sessions mentioned above, raises the question: how often *should* keys be changed?[15] This question proved to be *remarkably* hard to answer from basic Internet searches, including reading cryptography research papers and the *extensive* mailing list archives from development of key Internet protocols. Consequently, some parts of this document (annotated where appropriate) are derived from direct conversation with the principals.

There are two fundamental reasons for periodic rekeying:

1. Having a long string of data encrypted with the same key increases the probability of an attacker being able to find the key.

2. As a practical matter, reducing the amount of data that is exposed in the event that a key is broken is good stewardship of your data.

So changing the key “often” makes it both harder and less valuable for an attacker to find a key. That’s the tl;dr. Turning it into concrete, numeric recommendations is hard, but it looks like rekeying every half hour is pretty good, if your key generation rate isn’t high enough to do full one-time pad.

B. Concepts and Terminology

Cryptography is the science of secrets, and has a history going back several thousand years[16,17]. Modern practice with digital communications effectively began in the 1970s[18,19]. *Computer security* often involves cryptography, but is a far broader field tasked with ensuring the confidentiality, integrity and availability of data, computing and communication resources[20].

You probably all know that there are two major kinds of cryptography – symmetric key and asymmetric key, also known as public key. Due to the high costs of computation for public key, bulk data encryption is done using symmetric key. Symmetric encryption comes in two kinds, block ciphers and stream ciphers. These days pretty much everything seems to be block, but I’m not sure why.

Some additional terminology:

- cleartext: data that isn’t encrypted and isn’t really intended to be (sometimes confused with the below, even by me)
- plaintext: the original, unencrypted message
- ciphertext: the encrypted message
- integrity: the data hasn’t been tampered with
- confidentiality or privacy: the data hasn’t been disclosed to anyone unauthorized
- session keys: the keys used for one communication session
- forward secrecy: keeping your data secret in the future, esp. by building a cryptosystem that doesn’t reuse keys
- rekeying: changing the keys used in the middle of a session
- subkeys: keys used for a portion of an encryption process, derived from a subset of the bits of the session key
- cryptoperiod: the time that a specific key is authorized for use

Attacks on encrypted communications generally fall into one of three categories:

- unknown plaintext: the hardest problem; how do you recognize when you’ve found the message? With many but not all systems, failure will leave only unintelligible, random data, while success will produce words from the dictionary or other recognizable text.
- known plaintext: when an attacker knows what the plaintext corresponding to a particular ciphertext is, and attempts to find the key; not uncommon given the regularity of communications such as web browsing or email
- chosen plaintext: when the attacker can control the text to be encrypted, but obviously not the key; rarer than known plaintext, but can happen with small devices that a person may ”own” but not always completely control, or if the attacker partially controls some subset of resources, such as a related web server, or has compromised one or more hosts behind an encryption gateway

We also need these definitions:

- brute force/exhaustive search: checking every possible key, which of course is 2^n for an n-bit key, resulting in an expected hit time of half that number of trials; if you have a method that will find a key in substantially less than 2^{n-1} trials, you have ”broken” the cipher, even if your attack isn’t necessarily practical in the short run.
- pentesting (penetration testing)

And three mathematical definitions I know for algebra on the real numbers, but I’m a little fuzzy on in the bitwise, cryptographic context:

- linear function: I think in this context, $f(x, y)$ is a linear function of some bits if it involves only a linear addition of the inputs, and $f(0, 0) = 0$ (origin is preserved). Multiplication (AND) is disallowed? Importantly, $f(x_1 + x_2) = f(x_1) + f(x_2)$.
- affine function: same as a linear function, but an offset is allowed, such that $f(0, 0) = 1$ (origin isn’t necessarily preserved) (equivalent to a translation in a real space R^n).
- nonlinear function: A nonlinear function is one in which $f(x + y) \neq f(x) + f(y)$ for some values x and y. An affine function is one type of nonlinear function. I’m definitely fuzzy here...multiplication and arbitrary mappings are allowed?

II. ENCRYPTION

Let’s go phase by phase:

A. Authentication

The authentication phase can be accomplished using a pre-shared secret key and symmetric encryption of some message and response[21], or it can be done via asymmetric, public-key cryptography[22].

The best-known and most-used form of public key crypto is RSA, developed by Rivest, Shamir and Adelman (but also previously discovered by Cocks, of a British intelligence agency):

1. Pick two large primes, p and q, let $n = pq$.
2. Calculate $\phi(p, q) = (p - 1)(q - 1)$.
3. Pick e, prime relative to $\phi(p, q)$.
4. Calculate d, s.t. $de = 1 \bmod \phi(p, q)$, which we can call the inverse of e modulo $\phi(p, q)$.

The tuple $\langle n, e\rangle$ is now your public key, and d is your private key. You can publish $\langle n, e\rangle$ any way you like, including in the New York Times. To send you a message m, I calculate the ciphertext c,

$$c = m^e \bmod n \tag{1}$$

and send c to you. You can recover the message m by

$$m = c^d \bmod n. \tag{2}$$

That's all there is to it! Except that it's expensive, so we don't use it for every message. If I send you "Hi, Bob, it's Tuesday, let's use 42 as our key," using your public key and you reply, "Tuesday is beautiful, Alice, and 42 is fine," using my public key, we confirm that we are definitely Bob and Alice, known as authentication – assuming, of course, that you trust that the key $\langle n, e\rangle$ that you are holding really and truly belongs to me, and I haven't leaked it out somehow!

As you might guess from the publication of n, RSA is the one that's vulnerable to factoring larger integers, and hence of interest to spooks once Shor's algorithm came about.

Current EU recommendations are for 3072-bit RSA keys, recently (2018) upgraded from 2048, for "near-term protection" (up to ten years). They recommend an extraordinary 15360 bits for "long-term protection" (thirty to fifty years). [Cloudflare, Keylength.com, Ecrypt, Lenstra]

Finding prime numbers large enough to be used also requires a substantial amount of computation[23].

[called ECRYPT-CSA 2018, via Shigeya]

B. Key generation

The other algorithm we talk about a lot as being both important and vulnerable to Shor's factoring algorithm is Diffie-Hellman key exchange, which is used for creating the session key we will use for bulk data encryption. It's important that every session have its own separate encryption key; we don't want multiple conversations sharing the same key, for a lot of obvious reasons.

D-H works as follows:

1. Alice and Bob publicly agree on a modulus p and a base g (in cleartext is okay)
2. Alice and Bob each pick a secret random number a and b
3. Alice calculates $A = g^a \bmod p$,
   Bob calculates $B = g^b \bmod p$
4. Alice sends Bob A,
   Bob sends Alice B (in cleartext is okay)
5. Alice calculates $s = B^a \bmod p$,
   Bob calculates $s = A^b \bmod p$

Both Alice and Bob now have the same secret s, which they can use as an encryption key.

Note that, as-is, D-H is vulnerable to a man-in-the-middle attack, and so must be coupled with some form of authentication so that Alice and Bob each know that the other is who they say they are.

C. Bulk data encryption

D-H and RSA are pretty easy to understand, and known to be vulnerable to quantum computers, hence attract a lot of attention. The workhorse symmetric block ciphers for bulk encryption are actually much more complex mathematically, and hence harder to understand, but ultimately can be executed efficiently on modern microprocessors and dedicated chips.

A block cipher takes the data to be encrypted, and breaks it into fixed-size chunks called blocks. If the last block isn't full, it is filled out with meaningless data. We also take a key, and using the key perform mathematical operations on the data block. In a symmetric system, by definition, the decryption key is the same as the encryption key. Generally, the output block is the same size as the input block. There is no requirement that the block and key are the same size.

Ideally, the output block (the ciphertext) will look completely random: about 50% zeroes and 50% ones, regardless of input, and with no detectable pattern. That is, its entropy will be very high. Of course, it cannot be completely random, as it must be possible for the data to be decrypted by someone holding the appropriate key. A good cipher, however, will provide few clues to an attacker. A single bit's difference in either the key or the original plaintext should result in about half of the ciphertext bits being flipped, so that being "close" offers no guidance on a next step.

Thus, a symmetric key system's security is often linked to its key size; with k key bits, it should require, on average, 2^{k-1} trial decryptions to find the key and to be able to decrypt the message. We will discuss this further when we get to cryptanalysis.

Many encryption algorithms have been designed over the years, but two in particular are too important to ignore, so we will examine them: DES, the Data Encryption Standard, which is still in use in modified form but is primarily of historical interest now, and AES, the Advanced Encryption Standard, which is used for most communications today.

1. DES (the Data Encryption Standard)

DES, the Data Encryption Standard, was designed by IBM in the 1970s, consulting with the NSA. DES is a type of cipher known as an iterative block cipher, which breaks data into to blocks and repeats a set of operations on them. The operations in DES are called a Feistel network, after the inventor.

DES uses a 56-bit key, though products exported from the U.S. were limited to using 40 meaningful key bits [wikipedia/40-bit-encryption]. It was later upgraded to triple-DES, using three 56-bit key pieces and repeating DES three times, giving up to 168 bits of protection. But it's not just the key size that matters in a block cipher. The block size itself matters a great deal, as we'll see below. For DES, that block size is only 64 bits.

DES operates in sixteen rounds, each of which uses a 48-bit subkey generated from the original 56-bit key using a key scheduling algorithm. In each round, half of the block is tweaked and half initially left alone, then XORed with the tweaked half. The two halves are swapped before the next round.

The "tweaking" of the right half of the block is done by first expanding the 32 bits into 48 by replicating half of the bits, XORing with the 48-bit subkey, then dividing it into 6-bit chunks and pushing each chunk through one of eight substitution boxes, or S boxes. Each S box turns 6 bits into 4, using a lookup table defined as part of the algorithm (that is, this operation is not key-dependent). The S boxes are nonlinear (but not affine), which is the source of the true security of DES; if the S boxes were linear, breaking DES would be easy (or so I am told).

Decrypting DES is exactly the same operation as encrypting, except that the subkeys are used in reverse order.

Slightly more formally, the sequence of operations in a DES encryption is:

1. Apply initial permutation (IP) (a fixed operation)
2. For $i = 1$ to 16 do
   (a) divide the block into two 32-bit halves
   (b) expand the left half to 48 bits (a fixed operation)
   (c) calculate subkey i:
      i. split key into two 28-bit halves
      ii. rotate each half 1 or 2 bits (a fixed operation according to the key schedule)
      iii. select a subset of 48 bits (a fixed operation according to the schedule)
   (d) XOR subkey i with the left half of the block
   (e) split into eight 6-bit pieces
   (f) push each 6-bit piece through a $6 \rightarrow 4$ S-box
   (g) permute and recombine the eight 4-bit pieces (a fixed operation)
   (h) XOR the left half with the right half of the block
   (i) swap halves of the block
3. Apply the final permutation (FP) (a fixed operation)

The $6 \rightarrow 4$ S boxes are obviously inherently non-reversible, but the earlier expansion guarantees that ultimately no information is lost as the block passes through the entire network.

The success of cryptanalysis is often measured in terms of the number of rounds it can break in a multi-round cipher like DES. e.g., if DES were six rounds instead of sixteen, we could do yea much. Various attacks have been able to penetrate DES to different depths. Of course, the more straightforward approach is sheer brute force; as early as 1977, Diffie and Hellman published a critique in which they argued that brute force stood at the edge of feasible[24], and indeed in 1999, a special-purpose machine named Deep Crack, built by the Electronic Freedom Foundation, cracked a DES key. Further advances have made it possible even for dedicated hobbyists to crack.

Triple-DES (also called 3DES) has several modes of operation, but is usually used with three independent 56-bit keys, $K1$, $K2$, and $K3$, with encryption performed as $C = E_{K3}(D_{K2}(E_{K1}(P)))$ where P is the plaintext, E and D are the encryption and decryption operations, and C is the ciphertext.

DES was withdrawn as a standard in 2005, after having been replaced by AES in 2001, although the U.S. government still allows 3DES until 2030 for sensitive information.

2. AES (the Advanced Encryption Standard)

AES, the Advanced Encryption Standard, has replaced DES. It is a type of block cipher known as a substitution-permutation cipher. AES uses a block size of 128 bits, and a key size of 128, 192 or 256 bits. The key size affects the number of rounds of substitution-permutation, requiring 10, 12 or 14, respectively.

AES began life in 1997 when NIST announced a competition for a successor to DES, due to the problems with DES described earlier and later. Two Belgian cryptographers, Vincent Rijmen and Joan Daemen, submitted a proposal they dubbed Rijndael, which ultimately beat out fourteen other serious competitors to become, with some modifications, the Advanced Encryption Standard, published as the standard in 2001.

A substitution-permutation network, as you might guess, alternates substituting new bits for old ones with permuting (rearranging) the bits, conducted over a series of rounds. DES has some characteristics of a substitution-permutation network, but is not purely so. A well-designed s-p net will maximize Shannon's confusion (or the avalanche effect) and diffusion, which we will see shortly (Sec. III A 2).

AES begins by laying out the sixteen bytes in a 4×4 array in column major form: the first byte in the upper left, then the next byte below it, with the fifth byte returning to the top of the second column. Most operations work on rows, however, increasing the mixing of the bytes.

First, in the KeyExpansion phase, the round keys are created from the original key.

Next, the appropriate number of rounds is performed. Each round consists of the following steps:

	16-byte chunks	8,192-byte chunks
128-bit key	1.25 Gbps	1.44 Gbps
192-bit key	1.06 Gbps	1.18 Gbps
256-bit key	912 Mbps	1.01 Gbps

TABLE I. AES-CBC encryption rates on a Macintosh laptop with a 2.2GHz Intel Core i7 processor

1. SubBytes
2. ShiftRows
3. MixColumns
4. AddRoundKey

In the first round, AddRoundKey is performed before the other operations as well as after. In the last round, the MixColumns step is omitted.

The S box in SubBytes is a single, byte-level, fixed operation, $1:1$, unlike the DES S boxes, which are more complex and vary depending on the position within the block. The AES S box is nonlinear ($f(x+y) \neq f(x)+f(y)$) and was designed specifically to defeat both linear and differential cryptanalysis (which we will see later), but should you not trust the specific values they can be modified. The S box can be implemented as a straightforward 256-byte lookup table, with the input byte as the index, or as a simple bitwise matrix multiplication.

The ShiftRows operation rotates all rows but the first. The MixColumns operation is more complex, executed as a matrix multiplication using modulo-two integer arithmetic. These two operations maximize the diffusion of the original data. The AddRoundKey is a simple XOR of the 128-bit round key into the block.

AES is easily implemented in either software or hardware. Some modern CPUs include special instructions to accelerate AES encryption[25] , and even for those that don't, heavily optimized assembly language is achievable. In the absence of special hardware, encrypting each 128-bit block of data requires several hundred CPU instructions. Example encryption rates for a modern laptop are shown in Tab. I

Despite two decades of effort, there are no known attacks on AES that allow recovery of the key in substantially less time than brute force; the best attacks roughly halve that cost, at the cost of requiring tremendous amounts of storage.

3. *Limitations of Block Ciphers*

One obvious problem with the most straightforward application of a block cipher is that it's deterministic. If you just apply the transform to each block independently, it's easy to implement; this is known as electronic code book (EBC) mode. BUT: If you see the same ciphertext c in two different places in the message stream, you know that the two input blocks were the same! This leaks a huge amount of information to a sophisticated attacker, and is considered unacceptable.

One answer to this is to make the encryption slightly less deterministic by XORing in the ciphertext of the *previous* block into the current one before performing the encryption. This is cipher block chaining (CBC) mode, the standard mode of operation. (There are at least two more modes that I know nothing about.) CBC has the undesirable side effect of requiring encryption to be done serially, but attacks can be parallelized. Nevertheless, it's the most commonly used mode.

D. Other Tasks

⇒Crypto also used for signatures; integrity hashes (both signed ans unsigned); spread spectrum for secrecy, difficulty of triangulation and being hard to jam; challenge response rfid and smart credit cards.

⇒bitcoin here or elsewhere? Blockchain is an auditable communication channel[26].

⇒Internet security-oriented encryption not covered yet: ssh/scp, DNSSEC, BGPsec

E. Notes & References

To be added.

III. CRYPTANALYSIS

⇒Get Aono, Moriai or others to contribute here.

The general idea of trying to decode messages that are encrypted is known as cryptanalysis. Some of the planet's smartest people have worked on this for all of their lifetimes, building on work in progress since well before we were born, so my summary here is inevitably going to be superficial and perhaps wrong. Comments welcome; there is a lot of *bad* information about cryptography on the web, and I'd prefer to be corrected and make mine *good* information for the web.

(Somewhere, right now, maybe even as I type, employees of one or more of the world's major intelligence agencies are snickering at how naive my presentation here is. We don't know how much they really know, but we do know that e.g. the RSA public key cryptosystem and the technique of differential analysis were known to U.S. agencies for years, perhaps decades, before they became public.)

`https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#Known_attacks` says, "For cryptographers, a cryptographic "break" is anything faster than a brute-force attack – i.e., performing one trial decryption for each possible key in sequence (see Cryptanalysis). A break can thus include results that are infeasible with current technology." The page then goes on to list quite a number of attacks on AES, none considered practical yet.

Given how long people have been working on cryptography, naturally there are many techniques for attacking the ciphers. Here I'll mention just a couple of the modern techniques are known for what we might call "honest" attacks on block cipher cryptography, granting the assumption that the algorithm has no flaws and ignoring side channel attacks (such as measuring the power consumed during encryption) and attacks on infrastructure, people, etc. (some of which are known as "black bag", or burglary, attacks, and some of which are "rubber hose" attacks, extracting information from humans by coercion or torture). David Kahn referred to these latter attacks as "practical cryptanalysis" in his classic book, *The Codebreakers*.

The older one of the two is differential cryptanalysis, apparently discovered indepedently at least three times, publicly by Biham and Shamir in the late 1980s, in the mid-70s by IBM (where the goal of defending against the technique drove DES design decisions), and earlier by NSA. `https://en.wikipedia.org/wiki/Differential_cryptanalysis` The second is linear crypanalysis, discovered by Mitsuru Matsui in 1990[27]. `https://en.wikipedia.org/wiki/Linear_cryptanalysis`

(Other techniques that I know nothing about include integral, algebraic, and biclique.)

Before talking about those two in any detail, there's a more straightforward technique that leaks information about the plaintext, if not the key itself: the birthday paradox. But before we get into the cryptanalysis itself, let's look a little bit at how the designers built defenses into the block ciphers.

A. Playing Defense Using Math

1. Entropy

The single most important concept in cryptography – indeed, in all of information theory, and inarguably one of the most important in all of computer science – is Claude Shannon's *entropy*. The entropy is, roughly, the amount of disorder in a sequence of data, or the amount of information that must be transmitted to reconstruct the data. When the entropy of a cleartext message is high, it is hard to predict the next symbol (bit or byte or letter or floating point number); purely random data has very high entropy. When it is low, the cleartext has a skewed probability distribution such that, e.g., the letter 'e' or the number 0 is more common than other values.

Of course, because encryption tries to hide information, encrypted data looks as random as possible: it has very high entropy. Thus, an easy way to automate a test to see if we have successfully decrypted a message is to calculate the entropy of our provisionally decrypted plaintext; low entropy indicates a high probability that we have (at least partially) succeeded.

The entropy is defined as

$$H_2(X) = -\sum_{i=1}^{N} \frac{\mathrm{count}_i}{N} \log_2\left(\frac{\mathrm{count}_i}{N}\right) \tag{3}$$

where count_i is the number of times that the ith value appeared in our data, and N is the number of different values that show up in our data sequence.

Note that this definition of entropy does not take into account any long-range structure; it is purely based on the overall probability of the appearance of a given set of symbols. Thus, to this function, the strings 0101...01 and 000...01...111 have a bit-level entropy just as high as a string composed of exactly 50/50 randomly chosen 0s and 1s, although by applying a little intelligence we can easily reconstruct the former but not the latter (at least, not exactly). Many lossless data compression programs operate on stateful *sequences* of data to predict the next symbol. For example, the common Unix utilities gzip and compress use the Lempel-Ziv algorithm[28], which builds a dictionary of sequences as it goes, so that repetitive data and long runs are compressed effectively; this is especially good at sequences such as 0101...01 or 000...01...111.

(Taking still further advantage of brainpower, the ultimate in compression is measured using the *Kolmogorov complexity*: the length of the shortest program that will reproduce the sequence. This has very low complexity for anything highly regular, as well as seemingly irregular sequences such as the output of a pseudo-random number generator or the digits of π. I am not aware of any automated form for calculating the Kolmogorov complexity of an arbitrary sequence of data; in fact, it is known to be equivalent to the halting problem[29,30].)

Many other tests for randomness are important and are used to validate experimentally how completely an encryption system hides its data. If any pattern at all shows up in the ciphertext, that can be leveraged in an attack, and is an indication that the encryption scheme is poor.

This, obviously, is one of the motivations for the substitution phase in a substitution-permutation network; with permutation only the entropy of the original cleartext is preserved in the ciphertext, and in some cases reveals a great deal about the original data.

2. Diffusion, Confusion and the Avalanche Effect

Claude Shannon, in his seminal article on the mathematics of cryptography[31], defined two concepts he called *diffusion* and *confusion*[32]. In diffusion, information from the original plaintext message is spread across more than one symbol in the output ciphertext; the farther the information spreads, the better. Shannon defined *confusion* as making "the relation between...[the ciphertext] E and the...[key] K a very complex and involved one."

Feistel, who designed DES, called diffusion the *avalanche effect*. Webster and Tavares[33] defined the *strict avalanche criterion* (SAC), requiring that changing a single input bit flips each of the output bits with 50% probability.

The Handbook of Applied Cryptography[19] says the following (p. 20 in my edition):

> A substitution in a round is said to add *confusion* to the encryption process whereas a transposition [permutation] is said to add *diffusion*. Confusion is intended to make the relationship between the key and the ciphertext as complex as possible. Diffusion refers to rearranging or spreading out the bits in the message so that any redundancy in the plaintext is spread out over the ciphertext.

I haven't seen it written quite this directly (but then, I'm not that well read in crypto), but I think it's fair to say that confusion is achieved by the nonlinearity of the S-boxes.

These two concepts don't seem to figure prominently in the cryptography books and papers I have been reading, but it seems to me that they ultimately underly much of the process of cryptanalysis: spreading data broadly reduces the relationship exhibited by two ciphertexts even when the plaintexts are closely related, expanding the search space; and the nonlinearity means that even a large set of straightforward equations is not enough to simply and mathematically relate the input and output.

B. The Birthday Paradox

(Also called the pigeonhole principle or the hash collision problem.)

You're probably familiar with the basic idea of the birthday paradox: If you have n people in a room, what's the probability of two of them sharing a birthday? More concretely, how many people have to be in the room before the probability of two sharing a birthday exceeds 50%? It's a surprisingly small number, but it's not a true paradox in the logical sense. (This is also called the pigeonhole principle or hash collision probability.)

Modern ciphers are designed so that the ciphertext (output of the encryption) looks random; we can say that the ciphertext has very high entropy. If the block size is n bits, each of the 2^n possible bit combinations should be an equally likely result of encrypting a data block. (The encryption is deterministic, but the output looks random.)

If we monitor a long stream of encrypted data, there is some probability that we will find two blocks that have the same ciphertext. We call that a *collision*. If the two blocks were encrypted with the same key, we gain information about the plaintext.

(Side question: How can you *have* birthday paradox collisions of the ciphertext? Wouldn't that mean that the encryption process is lossy, and that it would be impossible to reliably *decrypt* that ciphertext back to both original blocks?

Answer: there is more information involved in both the encryption and decryption than just that specific block and the key. This is the cipher block chaining mentioned at the end of Sec. 1.)

In CBC mode, what you gain is the XOR of two plaintext blocks. If you get to choose which two, this could be incredibly valuable information; given that it's just two random blocks, it's less so, but not unimportant. As it happens, if block numbers i and j have the same ciphertext $c[i] = c[j]$, then you get the XOR of the plaintext of blocks $i-1$ and $j-1$.

I spent quite a bit of time working through the math of the birthday paradox, only to come back to one of the first sources I found: the 2016 SWEET32 paper by Bhargavan and Leurent. Sec. 2.2 of that paper has a compact description of the commonly-quoted square root limit, as well as why and how much to be conservative relative to that value.

If the block size is n bits, there are $N = 2^n$ possible ciphertexts, and if the number of blocks encrypted is the square root of that number, $2^{n/2}$, the probability of at least one collision is above 39%, which arises in the limit as the expression $1-1/e^{1/2}$. (See the appendix of these notes for some example calculations of this and the following to play around with.)

Assuming you consider a 39% chance of disclosing some information to be an unacceptably large probability, when should you rekey? That's the first-level answer to our ultimate question of when to rekey using QKD, which is of course back where we started this conversation. Say, for example, we want the probability of an attacker recovering a plaintext block using the birthday attack to be less than (for example) one in a billion.

If we have some power of two $D = 2^d$ ciphertext blocks, then the expected number of collisions is approximately 2^{2d-n-1}. For our one-in-a-billion probability, using $log_2(10^9) \approx 30$, we need to set up our session lifetime such that $2d - n - 1 < -30$, or $d < (n - 29)/2$.

Since DES has only a 64-bit block, we should set $d \leq 17$. That's a startlingly small number: $D = 2^{17}$ blocks and each block is eight bytes, so we should limit the use of a single key to one megabyte! An impractically small size.

If, on the other hand, you are okay with a disclosure probability of one in a million, we can raise that by a factor of a thousand and change keys once every gigabyte instead. But if you are trying to protect a 40Gbps link – a common backbone bandwidth today, and coming to the home in the foreseeable future – that still means changing keys once every 200msec or so!

Ah, but AES uses $n = 128$, a block size twice as large. Now we only need $d \leq 49$ for that one-in-a-billion probability. $D = 2^{49}$ times our block size of 16 bytes equals 8 terabytes, about 1,600 seconds on our 40Gbps link. So change the keys at least once every half hour, and you're good.

Keep in mind a few points:

1. what's disclosed here is not the key but is plaintext-related, but not even pure plaintext; however, the insanely smart and devious cryptanalysts can do amazing things with small amounts of data;

2. this depends only on the block size of a block cipher, not on the key length;

3. part of this arises due to the CBC (cipher block chain) mode, but ECB (electronic code book) mode is worse; there are other modes that I haven't looked at closely; and

4. given that there are still other attacks, minimizing the amount of data under any given encryption key is still good practice.

So let's look at a couple of the cryptanalysis techniques.

C. Differential Cryptanalysis

Biham and Shamir wrote a seminal paper[34], then an easy-to-read book[35] on their rediscovery of differential cryptanalysis. The goal of DC is to recover the key used for a session, so it is potentially far more serious than the birthday attack.

The book says, "An interesting feature of the new attack is that it can be applied with the same complexity and success probability even if the key is frequently changed and thus the collected ciphertexts are derived from many different keys." That's pretty alarming. This appears in a paragraph discussing chosen plaintext, so it may be restricted to that case. I infer from this that rather than needing the cumulative accretion of information, each trial is independent.

The attack works with either *chosen* plaintext (in which the attacker says, "Please encrypt this message for me, and give me the ciphertext,") or *known* plaintext (in which the attacker knows that the message is a repetition of "HEILHILTER", as figured prominently in the cracking ofo the Enigma machine in WWII; modern HTTPS connections and SMTP (email) connections have enough predictability in their content to provide a similar fulcrum for such a lever; see Sec. 2.x of these notes). There is a substantial difference in the complexity of the two attacks (see Tab. 2.1 in the book). Known plaintext takes *waaay* more trials to succeed.

The key idea is the construction of *differentials* (hence the name) from specific S boxes. Take two possible inputs to an S box, X_1 and X_2. We can assume the subkey has been XORed into both X_1 and X_2 already. $Y1$ and $Y2$ are the corresponding outputs of the S box. We know that if $X_1 = X_2$, then $Y_1 = Y_2$ and also $X_1 + X_2 = 0$ (again, here '+' is addition modulo two, or XOR).

For the total encryption algorithm, changing one bit in the input should result in about half the output bits changing. The S box should be similar; small changes in the input should mean large changes in the output. In fact, the S box is small enough that we can exhaustively analyze its inputs. We also know some rules that were chosen during the design phase, for example, changing *one* bit in the six-bit input should result in changes to at *two* bits in the four-bit output.

So a differential table for each S box is constructed by listing all 2^6 possible XORs $X_1 + X_2$, and collecting the stats on $Y_1 + Y_2$. From the differences found here, we can work backwards to find with "high" probability (slightly higher than completely random, at any rate) some characteristics of the outputs of the *previous* round.

The overall attack is performed by encrypting a bunch of randomly-chosen *pairs* of plaintexts (chosen as a pair; first a completely random one, then a second by XORing in a specific value) and comparing their ciphertexts until we find an output pair of ciphertexts that fit comfortably with the difference tables we have pre-computed for the S boxes. Repeat until we have built up some statistics about the right set of bits in the output, and from that we can take a probabilistic guess at the subkey in the last round. Based on that guess, we can narrow down the set of subkeys for the next-to-last round, rinse and repeat. It's still a computationally intensive, tedious process, but much less than brute force. Roughly, the probability of getting the ciphertext pairs we need is proportional to $1/p_D$, where p_D is the differential probability in the table we are looking for, which may be quite small. (This seems to me that keeping a history of things you've already tried would increase the probability of finding a pair you like, so I'm still puzzled by the assertion above that this works even if the key is being changed frequently.)

Ultimately, this attack is considered practical against ordinary DES. Biham and Shamir estimated (p. 8, p. 61) that DES and DES-like systems, even with the full sixteen rounds, could be broken using 2^{43} to 2^{47} chosen plaintext/ciphertext pairs. With 8-byte blocks, that's encryption of 64 terabytes up to a petabyte. If the system under attack can encrypt a 40Gbps link at line rate, that's only a few hours up to a couple of days of data.

Triple-DES would be much, much longer, but 3-DES is still considered obsolete due to the birthday paradox attack above. AES is stronger against DC, so this attack is of less help against properly-implemented, modern systems.

I know this is a very rough explanation; I have only a wobbly understanding of the process myself! The differential tables are pretty straightforward, once you understand them, but working from there to a full-on attack is a big jump.

D. Linear Cryptanalysis

Linear cryptanalysis (LC) is a known plaintext attack, developed by Mitsuru Matsui in 1990 after being inspired by differential cryptanalysis[27]. The key(!) idea is to build a linear approximation of an S box, allowing you to calculate possible keys in reverse more quickly.

If a cipher is good, every individual bit of the output should have a 50% probability of being 0 and a 50% probability of being 1. Likewise, there should be no obvious relationship between the input and output bits (e.g., "If the third input bit is 1, the seventh output bit is 0.") However, it is possible that some *combinations* of bits don't appear with equal probability. For example, the bit string composed of the first and third input bits and the second and fourth output bits should have all sixteen combinations 0000, 0001, ..., 1111 with equal probability, but perhaps there is a bias. If P_i is the ith input plaintext bit and C_i is the ith output ciphertext bit, we can construct an equation like

$L = P_1 + P_3 + C_2 + C_4$

(where '+' is modulo 2 addition, or XOR, here). We say that such a combination has a bias if the probability of L being 0 is noticeably different from 50%.

Such a combination is a *linear* combination of bits. It is known (by whom?) that if the S-boxes in our cipher are fully linear, then the cipher can be easily broken. Therefore, designers always use nonlinear S-boxes, but some bias such as this may be discoverable.

The basic idea of LC, then, is to find such sets of bits that exhibit some bias and use some algebra to recover a few bits of the subkey used in the last round of the cipher, rinse and repeat. The trick is to find the right relationships showing a detectable bias, as was done with differential analysis. This is done by examining the math of the S-boxes in detail; as far as I can tell, this phase of the operation is done by very smart humans.

If you can find a combination with a bias of ϵ, then you need $1/\epsilon^2$ plaintext-ciphertext pairs to find some bits of the subkey.

This is done by taking multiple expressions like the above, combining them using Matsui's "Piling Up Principle" where you track the biases multiplied together to make a linear approximation of the nonlinear S-box.

With this linear approximation, it is possible to find correlations between the output of the *next-to-last* round of the cipher and the original input qubits, from which it is possible to recover information about the *subkey* used in the last round of the cipher.

Ultimately, this doesn't give you complete information about the key or the plaintext, but substantially reduces the work needed to find the full key by guiding a search, rather than just testing keys at random.

Linear cryptanalysis is considered to be a very general technique, but I don't see extensive attempts to apply it to AES. Indeed, AES (which was developed in the 1990s from the proposed cipher known as Rijndael) was developed specifically with the idea of not being vulnerable to either DC or LC.

I found Heys' tutorial to be clear and helpful in understanding LC.

E. Known and chosen plaintexts in real systems

⇒Parts of this will make more sense after getting through the below. Maybe this section should be moved below the next section. Also, some pictures will definitely help here.

⇒This is fairly ad hoc, a set of rough notes I wrote up explaining some ideas without enough rigor. Would love to have someone with more sense than me look this section over.

Modern HTTPS (web) and SMTP (email) connections have a lot of predictability in their content, with commands like 'HELO' and 'HTTP/1.1' being standard parts of an exchange. In Sec. IV B, we'll see more detail about how this encryption is done using a protocol called Transport Layer Security, or TLS, but for the moment we'll only focus on the message contents and the fact that they are pretty predictable. Thus, it's reasonable to consider attacks on TLS to be known-plaintext attacks, and in fact there are cases where we can create chosen-plaintext attacks.

Consider, for example, the connection between your laptop and your email server, whether Gmail or your organization's server. Assume that I, an attacker, can send you email and can observe your encrypted connection to your server (perhaps I control a router somewhere between your server and your machine). I can send you an email message that contains, for example, the strings 0x000000000000000 (15 zero bytes in a row) and 0x000000010000000 (with a one in the middle). If your cipher block size is 8 bytes, as in DES, I know that one of the encrypted blocks will be all zeroes and one will have exactly one bit set, even if I have trouble controlling the exact position within the overall stream. I capture the ciphertext blocks, and compare them. This single-bit difference between two blocks helps me with the attack. All I have to do to execute a basic chosen-plaintext attack is to send you email and watch the resulting packets flow between your machines!

The success of such an attack requires a lot of assumptions about which parts of the entire process I can observe and which I can control, but using the principle of being conservative on security, assuming an attacker can force the choice of plaintext passed between two nodes through an encrypted connection is not unreasonable in today's richly interwoven distributed systems.

Especially for IPsec (Sec. IV A), there is another big vulnerability: one encrypted connection between two gateways (known as a *Security Association*, which we will see below) may carry data encrypted for a *bunch* of machines. So if an attacker manages to install a program on only one laptop (say, via email, or while you're sitting at Starbucks), they can cause your system to send out arbitrarily chosen packets that will cross the tunnel, so they can execute a chosen plaintext attack pretty easily. Since IPsec encrypts the whole packet, they may not be able to tell immediately which packets came from your laptop and which from a colleague's laptop, but that distinction is a relatively minor overhead.

Also, for every IP packet from your laptop to the email server passing through the IPsec tunnel, the IP header portion is going to be exactly the same, and its position in the encrypted stream is very easy to identify. This led to some of the decisions around the use of CBC, I believe; I'm not aware of any deeper features intended to further obscure the location of such predictable data.

In short, as a defender, you should work on the assumption that a noticeable fraction of your plaintext is known under benign circumstances, and that it's not all that hard for an attacker to mount a chosen plaintext attack.

F. Notes & References

Some of the references I used for this section:

The best source on the current state of the birthday paradox is Bhargavan and Leurent[36]. Other references include Miessler[37] and Wolfram Mathworld[38].

1. Howard M. Heys, "A tutorial on linear and differential cryptanalysis", undated but probably 1999ish? `http://www.engr.mun.ca/~howard/Research/Papers/ldc_tutorial.html`

2. Abdalla and Bellare, `https://link.springer.com/chapter/10.1007/3-540-44448-3_42` `http://cseweb.ucsd.edu/~mihir/papers/rekey.html` found via `http://cseweb.ucsd.edu/~mihir/`

That paper talks about differential/linear cryptanalysis and about the birthday paradox, saying block size k needs to be rekeyed every $2^{k/2}$ blocks.

Bellare et al, A concrete security treatment of symmetric encryption: analysis of the DES modes of operation abstract from STOC 1997 https://ieeexplore.ieee.org/abstract/document/646128 full paper at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.4734&rep=rep1&type=pdf` Focuses on DES, which was in the process of being superseded by AES even in 1997, but the content of the paper is valuable with respect to CBC. I found the paper a tough read when trying to figure out how to apply the equations.

IV. STANDARDS FOR ENCRYPTED COMMUNICATION

A. IPsec and the IETF

In this section, we return to the original question about rekeying that sparked this whole venture, and answer three questions:

1. What are the technical mechanisms in place for rekeying and effective use of encryption in the Internet standard security protocol IPsec?
2. What was known, at the time these protocols were developed, about the best practices for rekeying?
3. What are best practices today?

1. Background on IPsec, IETF and RFCs

The Internet Engineering Task Force (IETF) is where protocol specifications for the Internet come from. There is an entire Area within IETF (an "area" is the largest size organizational group in IETF, equivalent to a division of the American Physical Society, I would guess) dedicated to security, which charters many different working groups. Security is *MUCH, MUCH MORE* than cryptography, but an important area of work is developing the network protocols that allow real systems to use the cryptographic techniques discovered by the mathematicians. Moreover, theorists are inevitably naive about how much work it is to actually use their ideas. `https://trac.ietf.org/trac/sec/wiki`

One of the most important means of securing your communications is IPsec, which builds a "tunnel" inside of which ordinary IP packets can be carried transparent to their origin and destination (meaning your laptop and the server don't have to be be configured to handle the encryption; they deal in unmodified, unencrypted IP packets) but protected as they transit public networks.

IPsec is complex and has been updated many times. The Wikipedia page on it (which might be an easier entry point than the IETF indices, which are organized chronologically) lists over 40 standards-track documents, probably totaling over a thousand pages, some of which are outdated and some of which are still current. https://en.wikipedia.org/wiki/IPsec

Those documents are what are known as RFCs, or Request for Comments documents. They have different levels of authority, ranging from Experimental and Informational to Standard. Reaching Standard can take decades and numerous iterations as the working groups gradually converge on what works in the real world, intersecting with what people will actually implement and use, but protocols are often de facto standards long before reaching that platinum frequent flyer status.

2. IKE and IPsec

If you really want to dig into the key exchange protocol, RFC 7296 (Oct. 2014) is the most modern reference[39]. Unless you're actually manually configuring or implementing the stuff, you probably won't care about the differences between it and the older versions.

But you might want to start with RFC 4301 (Dec. 2005) (also a proposed standard), which is titled, "Security Architecture for the Internet Protocol"[40].

IPsec has a couple of modes, but let's stick to what's called tunnel mode. Two boxes, known as gateways, build one or more Security Associations (SAs). An SA describes which packets passing between the gateways are to be encrypted and how. Those that are encrypted are encrypted in their entirety (packet headers and all), and sent as the payload of another IP packet, to be decrypted at the far end. Tunnel mode is most often used to connect together via the Internet two networks (e.g., two offices of the same company) that are each considered to be relatively secure networks. The packets between computers in one network and those in the other network are encrypted only during their transit from gateway to gateway. Of course, these days, much (most?) data is also encrypted by the end hosts, especially for the two major applications of web and email, so much of the traffic in the tunnel will be double-encrypted.

The first SA created is the IKE SA itself, used only to carry the messages that govern the tunnel. The first exchange of messages negotiates some security parameters, and carries random nonces used to "add freshness" to the cryptographic exchange and the parameters to be used for the Diffie-Hellman key exchange. I believe this is where the preferred choice for the bulk encryption (3-DES v. AES v. whatever) is also negotiated. Since we have not yet established the tunnel, these messages are necessarily sent as plaintext.

A block of material called SKEYSEED is calculated independently by both ends using the nonces generated by both ends and the shared secret generated by the Diffie-Hellman exchange in the INIT. Building SKEYSEED involves the use of a pseudorandom function (PRF) also agreed upon...in the first exchange? I'm having trouble tracking where that's chosen.

SKEYSEED is used first to generate a key for the next message exchange, and then later to make keys for the Child SAs (below).

Next, there is an encrypted exchange that is used to authenticate the parties. The authentication may be via an RSA digital signature, a shared (symmetric) key message integrity code, or a DSS digital signature. In all three methods, each party signs a block of data using the secret, in a fashion that can be verified by the partner. (This could again be vulnerable to Shor's algorithm if it uses one of the public key methods, but keep in mind the messages containing this information are also encrypted; however, as we are just now authenticating, it's *possible* that, up to this point, the partner at the other end of this connection is not who they claim to be!)

The IKE SA is used to create Child SAs, which carry the actual traffic. The keys used for the Child SAs, therefore, are the obvious target for traffic-based attacks, though the real prize is the keys for the IKE SA. I'm having a hard time imagining how to mount an effective attack against the IKE SA.

The key material for the Child SA is generated via a complex mechanism involving a new nonce and the PRF previously specified. The initiator of the creation may also, optionally, specify that an entirely new Diffie-Hellman exchange be performed. I'm very unclear on how often that option is used in practice.

Each SA, whether IKE or Child, can (and should) have a lifetime. That lifetime can be specified in either seconds or in bytes that have been encrypted as they pass through the tunnel. Once the lifetime has expired, the two gateways must create a new Child SA with new keys. This ultimately is the heart of what we're looking for here: what is that recommended lifetime today, and what should it be in the light of quantum computing?

3. Digging into the Cryptanalysis of IPsec

What is the recommended lifetime of an IPsec Security Association today? This is the question that has proven to be so hard to answer, and that has led me wandering all across the web. Probably the most relevant source is, naturally, the mailing list where most of the design work is documented.

One early, relevant message (`https://mailarchive.ietf.org/arch/msg/ipsec/_dSgUvW6WiUFvw5aoyu7rJo5Kgc`) from 08 March 1995 carries the quote:

```
"I think 2^32 is a better bound than 2^43, at least for certain modes
of DES. For instance, after 2^32 blocks in CBC mode, you expect to see
two identical ciphertext blocks, say c[i] and c[j]; the difference
between their predecessors will match the difference between the
corresponding plaintext blocks, i.e.,

p[i] xor p[j] = c[i-1] xor c[j-1]

Information thus starts to leak after 2^32 blocks (square root of the
message space). I would recommend 2^32 blocks as the limit for the
lifetime of a key, and that takes care of the 2^43/2^47 attacks as
well."
```

referring, although not by name, to both the birthday paradox and the differential cryptanalysis limits discussed above. Keep in mind that at 2^{32} blocks, we are at a 39% probability of there being at least one ciphertext collision revealing some information.

Searching the archives for ”birthday” also turned up some relevant messages, e.g. the relatively recent (21 April 2015) message `https://mailarchive.ietf.org/arch/msg/ipsec/T1woQuwh1Ccoz6fWWFDBETBllaY` quoting the earlier message `https://mailarchive.ietf.org/arch/msg/ipsec/De46HNR1h4lGXZnxlIVpNSfs5p0`

```
"> I think the main problem with 3DES is not that it is significantly slower
> than AES, but that it has blocksize of 64 bits, that is considered
> loo small for high-speed networks, when the possibility of birthday attack
> leads to necessity to frequently rekey.

It's hard to make that case. The blocksize is 64 bits. So it's prudent
to not use more than, say, a billion blocks. A billion blocks is 64
Gb. There are very few real tunnels that run that kind of throughput
in under a minute. OTOH it's no problem at all to run a CreateChildSA
every minute, or even every five seconds. So I think there are very
few cases that *can't* use 3DES."
```

This is interesting, particularly given its newness. The author (Yoav Nir, one of the long-time leaders of the IPsec community) considers 3DES plus very frequent rekeying to be sufficient, at least for some use cases, and it's important for backwards compatibility. However, in a slightly earlier (2012) exchange on the mailing list, David McGrew (another key IPsec person) and Nir covered the same issue, with McGrew arguing that no more than 50 megabytes should be encrypted with the same key even using 3DES, due to the birthday paradox. McGrew went so far as to write up a 16-page analysis posted on the Cryptology preprint server[41].

4. IPsec, Quantum Computing and QKD

⇒Papers on actually integrating QKD into classical crypto[42,43].

⇒I have two major QKD reviews in my stacks somewhere, haven't found either yet...

(This one is just a placeholder, but there is surprisingly little *documented* about this kind of work. Even e.g. the publications describing the recent Chinese QKD satellite experiment[44] say, "We used AES plus QKD," or, "We used QKD-generated keys as a one-time pad," but they tell you nothing about the *network* or *transport* layer protocols used, so who knows? Might be documented elsewhere on the web, or described in talks, but so far I've failed to track down anything authoritative.)

Chip Elliott was the first to modify IPsec to work with QKD, as far as I'm aware, described in a 2003 SIGCOMM paper[45]. Shota Nagayama drafted and implemented a more formal protocol modification for his bachelor's thesis working with me, back in 2010, but we failed to push that from Internet Draft to RFC[46].

⇒Also[47].

(More to come here, eventually, on both the use of QKD with protocols such as IPsec and on how effective quantum computers will or won't be at breaking communication security. Some of this will also be covered in Section 5.)

B. TLS/SSL and cryptography

Here we want to answer the same three questions we had above:

1. What are the technical mechanisms in place for rekeying and effective use of encryption in TLS?
2. What was known, at the time this protocol was developed, about the best practices for rekeying?
3. What are best practices today?

but this section will be much shorter, as I know so much less about TLS (despite the fact that it is the most important use of encryption on the Internet today).

The current standard for Transport Layer Security, or TLS, is RFC8446[48], published in 2018. It specifies version 1.3 of the protocol. The main document itself is only 160 pages, not bad for something so complex and important. (The 1.2 spec was only about 100 pages, so it has grown substantially.)

...oh, what is TLS? It's the protocol that HTTPS runs over. It's the successor to SSL, the Secure Sockets Layer, which was the first way to encrypt web browsing. TLS originally built on top of a reliable transport protocol, such as TCP, though a later adaptation[49] lets it run over a datagram protocol. We'll only worry about running it over TCP here. TLS provides privacy, by encrypting all of your web browsing traffic for a particular connection. It also provides data integrity, using a MAC (message authentication code) when necessary. (IPsec also does both, but we ignored that above.)

1. TLS Records and Basic Limits

TLS consists of one primary protocol: the TLS Record Protocol. On top of the TLS Record Protocol sit four protocols, one of which (the application data protocol) is used for the bulk data encryption, and the TLS Handshake Protocol, which utilizes public key cryptography to establish identity, securely creates a shared secret to be used for the bulk data encryption, and makes sure that this part of the process can't be modified by an attacker. A third one of interest to us is the change cipher spec protocol.

A record in the record layer has a maximum size of 16KB ($2^{14}+1$, actually).

There is, as best I can tell, no official limit on the key lifetime or on the number of bytes that can be pushed through a single TLS except the limit on record sequence numbers of $2^{64}-1$. Combined with the max record size, that's 2^{78} bytes, or 256 exabytes, which is a bloody lot of data. So, if the lifetime of a session needs to be limited, it has to be done by breaking the connection and renegotiating. Apparently, adding a key renegotiate feature was considered for TLS 1.3, but doesn't seem to have been included.

Luykx and Paterson wrote up a short (8 page) summary recommending limits for TLS with different cryptosystems. (My copy is dated Aug. 2017, but the paper was referred to in Apr. 2016.) `http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf` Unfortunately, that paper has good references but doesn't go through the complete reasoning, so going one level deeper in the reading is really necessary.

In April 2016, Mozilla moved to limit the length of a connection, based on those results: `https://bugzilla.mozilla.org/show_bug.cgi?id=1268745` They set AES-CBC to about $2^{34.5}$, or about 24 billion records, about 400 terabytes max. That should give a probability of 2^{-57} of "ciphertext integrity [being] breached", though I'm a little unclear on exactly what that means here – is this just birthday bound leaking some plaintext, or is this compromise of the entire session key? Figuring that out will take more digging, since they express things differently than most of the resources I used above.

2. Keying and Rekeying

⇒how the initial key is generated

⇒add more here on chains of trust for certificates, and how we make sure that an attacker can't cause a downgrade of security relative to what both ends really want.

⇒how rekeying is handled

3. *Other Attacks on TLS*

One startling and potentially useful document is RFC7457, ”Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)”[50]. It details a couple of dozen attacks on TLS, most having to do with protocol implementation issues such as failing to correctly verify all of the information you are given. Some of them leak information or allow a connection to be take over, others allow an attacker to downgrade the security negotiation so that a later attack on the cipher has a higher chance of success.

One attack this RFC points out is `http://www.openssl.org/~bodo/tls-cbc.txt`, which (among other things) describes a flaw in how TLS Record Protocol records are concatenated. Originally, TLS kept the ciphertext of the last block of a record to use as the initialization vector (IV) of the first block of a new record, but this allows an attacker who can adaptively choose plaintexts to more completely choose the text being encrypted. The fix is straightforward (toss in an extra block of nonsense data before starting the new record), and current versions of TLS aren't vulnerable.

4. *TLS and QKD*

Alan Mink, Sheila Frankel and Ray Perlner worked on integrating TLS with QKD, a full decade ago[43].

⇒Performance requirement: interactivity means a few seconds or less; 300ms per click, is it?

C. Other Internet Security Protocols

⇒Get Woolf to contribute here.

⇒Internet security-oriented encryption not covered yet: ssh, DNSSEC, BGPsec

D. WiFi

⇒Get yume or maybe koluke to contribute here?

E. Cellular Phones

⇒Who can contribute here?

F. Notes & References

To be filled in eventually.

V. ATTACKING CLASSICAL CRYPTOGRAPHY USING QUANTUM COMPUTERS

Naturally, my own interest in writing these notes stems from my experience in quantum computing and quantum networking. The previous sections dealt primarily with classical attacks, with a little bit of quantum networking can be integrated with classical thrown into each section. Here, let's look at the attacks on classical cryptography using quantum computers.

A. Shor 'Nuff

Obviously, the main thing we're talking about here is Shor's algorithm. I have not been privy to the conversations of cryptographers in creating post-quantum crypto, though we'll take a short look at that below. But there are very few people in the world who understand better than I do what it takes to actually run the full version of Shor's algorithm on a potentially real machine.

A full implementation of Shor's algorithm consists of two quantum phases: modular exponentiation, followed by the quantum Fourier transform. (I'm assuming you're familiar with the main ideas behind Shor, how it uses interference to build patterns in the quantum register that can reveal the period of a function that allows us to find the factors of a large semi-prime number.)

The key insight may be the behavior of the QFT, but the bulk of the execution time will be in the modular exponentiation phase. This is actually one of the areas that I worked on in my Ph.D. thesis[51]. Some of the important parts of this were published in Physical Review A[52].

In my thesis there is a plot that I think is useful, which we updated and published in our Communications of the ACM article[53].

We worked out detailed performance estimates, including hardware requirements and quantum error correction, in some of our papers[54,55].

There were other, contemporary important papers on how to implement Shor, including Fowler et al.[56], who discussed Shor on a linear array, a few months ahead of my own Phys. Rev. A paper on the same topic.

We both built on important, early work by Vedral, Barenco and Ekert (VBE)[57] and by Beckman, Chari, Devabhaktuni and Preskill (BCDP)[58].

If you are looking to broaden your reading on implementations of Shor's algorithm, the following are useful:

Pavlidis and Gizopoulous found an efficient division algorithm, accelerating the math[59].

Roetteler, Steinwandt focused on the applicability of Shor's algorithm to elliptic curve. I like the paper, except I think their survey of related research could have been better. `https://arxiv.org/abs/1306.1161` and Roetteler, Naehrig, Svore, Lauter: `https://arxiv.org/abs/1706.06752`

Gidney and Ekera submitted to the arXiv a paper on factoring a 2048-bit number using 20 million noisy quibits. In this paper, one of their techniques is arithmetic ”windowing” which is essentially identical to one of the techniques I proposed in my 2005 Phys. Rev. A paper. `https://arxiv.org/abs/1905.09749`

May and Schliper also recently uploaded a paper on period finding with a single qubit. `https://arxiv.org/abs/1905.10074`

Ekeraa and Hastad also proposed a new period-finding algorithm variant, in 2017. `https://link.springer.com/chapter/10.1007/978-3-319-59879-6_20` `https://arxiv.org/abs/1702.00249`

Smolin, Smith, Vargo wrote something of a warning about inferring too much from very simple demonstrations on a few qubits. `https://www.nature.com/articles/nature12290`

Here is one early paper on how to assess errors in Shor's algorithm, by Miquel, Paz and Perazzo[60].

Chuang et al., also published an early examination of decoherence in factoring[61].

Among random things, there is Dridi and Alghassi, factoring on D-Wave; I don't think this paper tells us very much about factoring at scale[62].

That's more than enough for now, since it isn't really the focus of what we're after here, anyway.

B. Grover's amplifier

When learning about linear cryptanalysis (above), I naturally wondered I wonder how hard it would be to set up amplitude amplification for the bias in LC. The bias we are looking for is certainly small.

It turns out people have addressed that question: `https://arxiv.org/abs/1510.05836` `https://link.springer.com/article/10.1007/s11128-015-0983-3` `https://link.springer.com/chapter/10.1007/978-3-662-48683-2_5`

One new one I hadn't seen before: Bernstein, Biasse, Mosca on low-resource quantum factoring, using Grover's algorithm and NFS: `https://research.tue.nl/en/publications/a-low-resource-quantum-factoring-algorithm` `https://cr.yp.to/papers/grovernfs-20170419.pdf`

C. Notes & References

To be filled in eventually, mostly by moving the existing parts of this section into this subsection!

VI. POST-QUANTUM CRYPTOGRAPHY

⇒somebody besides me should write this. Moriai-san? Aono-san?

⇒Jon Dowling (RIP)'s opinions here were strong. He believed that post-quantum crypto is fundamentally impossible, that all of the interesting asymmetric problems useful for authentication and key generation will ultimately fall to quantum algorithms.

⇒Aono: lattice crypto is very attractive because a) it reduces to shortest vector or other problems, and b) implementation is easy, basically a matrix times a vector[63].

⇒learning w/ errors $As + e = t \bmod q$ candidate for post-quantum crypto

⇒(n.b.: cocori created a ipynb, but misunderstood the size of the matrix necessary)

Post-quantum cryptography is the attempt to find a public key cryptosystem that is resistant to quantum computing, Shor's algorithm in particular.

There is enough interest in this that the Wikipedia pages are essentially extensive catalogs: `https://en.wikipedia.org/wiki/Post-quantum_cryptography` `https://en.wikipedia.org/wiki/Post-Quantum_Cryptography_Standardization`

An official-looking site: `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization`

One recent blog posting of some use: `https://blog.trailofbits.com/2018/10/22/a-guide-to-post-quantum-cryptography/`

A very recent blog posting on teaching crypto in the post-quantum crypto age: `https://news.ncsu.edu/2019/06/teaching-next-generation-cryptosystems/` which builds on a conference presentation on the course they created: `https://dl.acm.org/citation.cfm?id=3317994` which goes into a lot of detail on crypto hardware.

A survey from a decade ago: `https://www.nist.gov/publications/quantum-resistant-public-key-cryptography-survey?pub_id=901595`

Also in 2009, there was a book, which I'm sure is almost entirely outdated by now: `https://www.springer.com/jp/book/9783540887010`

A. Notes & References

To be filled in eventually, mostly by moving the existing parts of this section into this subsection!

VII. POLICY AND CONCLUSIONS

⇒Get Farber to contribute here (and elsewhere).

⇒Add a few paragraphs here on how national and corporate policy drive, and are driven by, advances in crypto, and in consequence quantum.

ACKNOWLEDGMENTS

This document began as a very long email to a group researchers, following up on a question from Renato Renner about cryptographic key lifetimes at the Network Games, Tropical Topology and Quantum Communication Workshop in Berlin in 2019. Thanks to Jens Eisert for inviting me.

Thanks to Darrell Piper for insight into the early discussions of IKE and IPsec.

I used to work on a box known as an IPsec gateway, but I was the OS guy, not the cryptographer; Darrell Piper, Dan Harkins and Dave Kashtan did most of the crypto work. I am also a board member of the WIDE Project, which through its KAME Project created the Racoon software that was an important early implementation of the key exchange protocol for IPsec, with much of that work done by Shoichi Sakane. Much of what I know about IPsec and other security was just incidental radiation from them, or gained from the security-oriented papers we had to read when I did my master's at USC back before the Englightenment.

VIII. REFERENCES

[1] P. W. Shor, "Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer," SIAM J. Comp. **26**, 1484–1509 (1997), http://arXiv.org/quant-ph/9508027.

[2] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in *Proc. IEEE International Conference on Computers, Systems, and Signal Processing* (IEEE, 1984) pp. 175–179.

[3] C. H. Bennett, G. Brassard, and N. D. Mermin, "Quantum cryptography without Bell's theorem," Phys. Rev. Lett. **68**, 557–559 (1992).

[4] A. Ekert, "Quantum cryptography based on Bell's theorem," Physical Review Letters **67**, 661–663 (1991).

[5] H.-K. Lo, M. Curty, and B. Qi, "Measurement-device-independent quantum key distribution," Phys. Rev. Lett. **108**, 130503 (2012).

[6] F. Xu, M. Curty, B. Qi, and H.-K. Lo, "Measurement-device-independent quantum cryptography," IEEE Journal of Selected Topics in Quantum Electronics **21**, 148–158 (2015).

[7] U. Vazirani and T. Vidick, "Fully device independent quantum key distribution," Commun. ACM **62**, 133–133 (2019).

[8] V. Scarani, H. Bechmann-Pasquinucci, N. J. Cerf, M. Dušek, N. Lütkenhaus, and M. Peev, "The security of practical quantum key distribution," Rev. Mod. Phys. **81**, 1301–1350 (2009).

[9] P. W. Shor and J. Preskill, "Simple proof of security of the BB84 quantum key distribution protocol," Phys. Rev. Lett. **85**, 441–444 (2000).

[10] S. Kimmel and S. Kolkowitz, "No-go bounds for quantum seals," Phys. Rev. A **100**, 052326 (2019).

[11] H. Buhrman, N. Chandran, S. Fehr, R. Gelles, V. Goyal, R. Ostrovsky, and C. Schaffner, "Position-based quantum cryptography: Impossibility and constructions," SIAM Journal on Computing **43**, 150–178 (2014).

[12] M. Ben-Or and A. Hassidim, "Fast quantum Byzantine agreement," in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing* (ACM, 2005) pp. 481–485.

[13] C. Crep̀eau, D. Gottesman, and A. Smith, "Secure multi-party quantum computation," in *Proc. Symposium on Theory of Computing* (ACM, 2002).

[14] J. Eisert, D. Hangleiter, N. Walk, I. Roth, D. Markham, R. Parekh, U. Chabaud, and E. Kashefi, "Quantum certification and benchmarking," arXiv preprint arXiv:1910.06343 (2019).

[15] This question, in fact, prompted the development of this entire document.

[16] D. Kahn, *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet*, 2nd ed. (Simon and Schuster, 1996).

[17] S. Singh, *The Code Book* (Doubleday New York, 1999).

[18] B. Schneier, *Applied Cryptography*, 2nd ed. (J. Wiley, 1996).

[19] A. J. Menezes, J. Katz, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography* (CRC press, 1996).

[20] M. A. Bishop, *Computer Security: Art and Science*, 2nd ed. (Addison-Wesley Professional, 2018).

[21] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," Commun. ACM **21**, 993–999 (1978).

[22] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM **21**, 120–126 (1978).

[23] See `https://en.wikipedia.org/wiki/Primality_test`.

[24] W. Diffie and M. E. Hellman, "Special feature: exhaustive cryptanalysis of the NBS data encryption standard," Computer **10**, 74–84 (1977).

[25] On recent Intel processors the instructions AESDEC, AESDECLAST, AESENC, AESENCLAST, AESIMC, and AESKEYGENASSIST are custom-created for AES, and PCLMULQDQ is a more generic instruction for lock-oriented encryption.

[26] S. Suzuki and J. Murai, "Blockchain as an audit-able communication channel," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2 (IEEE, 2017) pp. 516–522.

[27] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Workshop on the Theory and Application of of Cryptographic Techniques* (Springer, 1993) pp. 386–397.

[28] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," IEEE Trans. Inf. Theory **IT-24**, 530–6 (1978).

[29] G. J. Chaitin, "A theory of program size formally identical to information theory," Journal of the ACM (JACM) **22**, 329–340 (1975).

[30] G. J. Chaitin, A. Arslanov, and C. Calude, "Program-size complexity computes the halting problem," Tech. Rep. CDMTCS-008 (Department of Computer Science, The University of Auckland, New Zealand, 1995).

[31] C. E. Shannon, "A mathematical theory of communication," (1945).

[32] `https://en.wikipedia.org/wiki/Confusion_and_diffusion`.

[33] A. F. Webster and S. E. Tavares, "On the design of s-boxes," in *Advances in Cryptology — CRYPTO '85 Proceedings*, edited by H. C. Williams (Springer Berlin Heidelberg, Berlin, Heidelberg, 1986) pp. 523–534.

[34] E. Biham and A. Shamir, "Differential cryptanalysis of des-like cryptosystems," Journal of CRYPTOLOGY **4**, 3–72 (1991).

[35] E. Biham and A. Shamir, *Differential cryptanalysis of the data encryption standard* (Springer Science & Business Media, 1993).

[36] K. Bhargavan and G. Leurent, "On the practical (in-) security of 64-bit block ciphers: Collision attacks on http over tls and openvpn," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016) pp. 456–467, see also `https://sweet32.info/`.

[37] `https://danielmiessler.com/study/birthday_attack/`.

[38] `http://mathworld.wolfram.com/BirthdayAttack.html`.

[39] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet key exchange protocol version 2 (ikev2)," STD 79 (RFC Editor, 2014) `http://www.rfc-editor.org/rfc/rfc7296.txt`.

[40] S. Kent and K. Seo, "Security architecture for the internet protocol," RFC 4301 (RFC Editor, 2005) `http://www.rfc-editor.org/rfc/rfc4301.txt`.

[41] D. McGrew, "Impossible plaintext cryptanalysis and probable-plaintext collision attacks of 64-bit block cipher modes," Cryptology ePrint Archive, Report 2012/623 (2012), `https://eprint.iacr.org/2012/623`.

[42] R. Alléaume, C. Branciard, J. Bouda, T. Debuisschert, M. Dianati, N. Gisin, M. Godfrey, P. Grangier, T. Länger, N. Lütkenhaus, C. Monyk, P. Painchault, M. Peev, A. Poppe, T. Pornin, J. Rarity, R. Renner, G. Ribordy, M. Riguidel, L. Salvail, A. Shields, H. Weinfurter, and A. Zeilinger, "Using quantum key distribution for cryptographic purposes: A survey," Theoretical Computer Science **560, Part 1**, 62 – 81 (2014), theoretical Aspects of Quantum Cryptography, celebrating 30 years of {BB84}.

[43] A. Mink, S. Frankel, and R. Perlner, "Quantum key distribution (QKD) and commodity security protocols: Introduction and integration," International Journal of Network Security & Its Applications (IJNSA) **1** (2009).

[44] S.-K. Liao, W.-Q. Cai, J. Handsteiner, B. Liu, J. Yin, L. Zhang, D. Rauch, M. Fink, J.-G. Ren, W.-Y. Liu, Y. Li, Q. Shen, Y. Cao, F.-Z. Li, J.-F. Wang, Y.-M. Huang, L. Deng, T. Xi, L. Ma, T. Hu, L. Li, N.-L. Liu, F. Koidl, P. Wang, Y.-A. Chen, X.-B. Wang, M. Steindorfer, G. Kirchner, C.-Y. Lu, R. Shu, R. Ursin, T. Scheidl, C.-Z. Peng, J.-Y. Wang, A. Zeilinger, and J.-W. Pan, "Satellite-relayed intercontinental quantum network," Phys. Rev. Lett. **120**, 030501 (2018).

[45] C. Elliott, D. Pearson, and G. Troxel, "Quantum cryptography in practice," in *Proc. SIGCOMM 2003*, ACM (ACM, 2003).

[46] S. Nagayama and R. Van Meter, "IKE for IPsec with QKD," (2014), internet Draft, draft-nagayama-ipsecme-ipsec-with-qkd-01; expired April 30, 2015.

[47] Y. TANIZAWA, R. TAKAHASHI, S. Hideaki, A. R. DIXON, and S. KAWAMURA, "A secure communication network infrastructure based on quantum key distribution technology," IEICE TRANSACTIONS on

Communications **99**, 1054–1069 (2016).
[48]E. Rescorla, "The transport layer security (tls) protocol version 1.3," RFC 8446 (RFC Editor, 2018).
[49]E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," RFC 6347 (RFC Editor, 2012) `http://www.rfc-editor.org/rfc/rfc6347.txt`.
[50]Y. Sheffer, R. Holz, and P. Saint-Andre, "Summarizing known attacks on transport layer security (tls) and datagram tls (dtls)," RFC 7457 (RFC Editor, 2015) `http://www.rfc-editor.org/rfc/rfc7457.txt`.
[51]R. D. Van Meter III, *Architecture of a Quantum Multicomputer Optimized for Shor's Factoring Algorithm*, Ph.D. thesis, Keio University (2006), available as arXiv:quant-ph/0607065.
[52]R. Van Meter and K. M. Itoh, "Fast quantum modular exponentiation," Physical Review A **71**, 052320 (2005).
[53]R. Van Meter and C. Horsman, "A blueprint for building a quantum computer," Communications of the ACM **53**, 84–93 (2013).
[54]R. Van Meter, T. D. Ladd, A. G. Fowler, and Y. Yamamoto, "Distributed quantum computation architecture using semiconductor nanophotonics," International Journal of Quantum Information **8**, 295–323 (2010).
[55]N. C. Jones, R. Van Meter, A. G. Fowler, P. L. McMahon, J. Kim, T. D. Ladd, and Y. Yamamoto, "Layered architecture for quantum computing," Phys. Rev. X **2**, 031007 (2012).
[56]A. G. Fowler, S. J. Devitt, and L. C. Hollenberg, "Implementation of Shor's algorithm on a linear nearest neighbor qubit array," Quantum Information and Computation **4**, 237 (2004).
[57]V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," Phys. Rev. A **54**, 147–153 (1996), http://arXiv.org/quant-ph/9511018.
[58]D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill, "Efficient networks for quantum factoring," Phys. Rev. A **54**, 1034–1063 (1996), http://arXiv.org/quant-ph/9602016.
[59]A. Pavlidis and D. Gizopoulos, "Fast quantum modular exponentiation architecture for Shor's factoring algorithm," Quantum Info. Comput. **14**, 649–682 (2014).
[60]C. Miquel, J. Paz, and R. Perazzo, "Factoring in a dissipative quantum computer," Physical Review A **54**, 2605–2613 (1996).
[61]I. L. Chuang, R. Laflamme, P. Shor, and W. H. Zurek, "Quantum computers, factoring and decoherence," Science **270**, 1633–1635 (1995).
[62]R. Dridi and H. Alghassi, "Prime factorization using quantum annealing and computational algebraic geometry," Scientific Reports **7**, 43048 (2017).
[63]O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," J. ACM **56** (2009), 10.1145/1568318.1568324.
[64]`http://rosettacode.org/wiki/Entropy`.

A. Not yet cited

8.1 IPsec:

`https://en.wikipedia.org/wiki/Internet_Security_Association_and_Key_Management_Protocol#Implementation` `https://en.wikipedia.org/wiki/Internet_Key_Exchange` `https://tools.ietf.org/html/rfc7296(IKEv2)` `https://en.wikipedia.org/wiki/IPsec` `https://trac.ietf.org/trac/sec/wiki(SecurityArea)` `https://tools.ietf.org/html/rfc2407(ddp)` `https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_(CBC)` `https://tools.ietf.org/html/rfc3602` (Frankel)

Additional IKE references:

ISAKMP: `https://en.wikipedia.org/wiki/Internet_Security_Association_and_Key_Management_Protocol` `http://www.racoon2.wide.ad.jp/w/` `http://www.kame.net/racoon/` ¡our URL¿

AES-CBC use in IPsec: `https://tools.ietf.org/html/rfc3602`

NIST key length recommendations: `https://www.keylength.com/en/4/` `https://www.keylength.com/en/`

Lenstra, Verheul, Selecting Cryptographic Key Sizes, J. Cryptology (2001),. 14:255-293 `https://infoscience.epfl.ch/record/164526/files/NPDF-22.pdf`

IP Security Maintenance and Extensions (ipsecme) Working Group `https://datatracker.ietf.org/wg/ipsecme/about/`

IPsec mailing list archives: `https://mailarchive.ietf.org/arch/browse/ipsec/` Searching that for "lifetime" resulted in (as of 2019/6/19) 1,018 email messages, the oldest of which was from July, 1993, and doesn't seem relevant.

8.2 General crypto:

`https://en.wikipedia.org/wiki/Advanced_Encryption_Standard` `https://brilliant.org/wiki/rsa-encryption/` `https://en.wikipedia.org/wiki/RSA_(cryptosystem)` `https://en.wikipedia.org/wiki/Forward_secrecy`

And of course the books

Schneier, Applied Cryptography `https://www.schneier.com/books/applied_cryptography/` Schneier, Ferguson, Kohno, Cryptography Engineering `https://www.schneier.com/books/cryptography_engineering/` Menezes, van Oorschot, Vanstone, Handbook of Applied Cryptography `http://cacr.uwaterloo.ca/hac/` (all chapters are available as free PDF downloads)

8.3 Cryptanalysis:

`https://en.wikipedia.org/wiki/Differential_cryptanalysis` `https://en.wikipedia.org/wiki/Linear_cryptanalysis` `https://www.cs.rit.edu/~ib/Classes/CS482-705_Winter10-11/Slides/crypto_lc.pdf` `http://www.engr.mun.ca/~howard/Research/Papers/ldc_tutorial.html` `http://www.ciphersbyritter.com/RES/LINANA.HTM` (a lit survey on LC; rather ad hoc)

8.4 Key lifetime & length:

`https://www.cryptomathic.com/news-events/blog/exploring-the-lifecycle-of-a-cryptographic-key-` `https://searchsecurity.techtarget.com/definition/cryptoperiod` `https://www.keylength.com/en/4/` `https://arxiv.org/abs/quant-ph/0306078` `https://royalsocietypublishing.org/doi/10.1098/rspa.2004.1372` `https://www.physics.utoronto.ca/research/quantum-optics/cqiqc_events/jean-christian-boileau-tba` `https://learningnetwork.cisco.com/`

thread/25765 http://cseweb.ucsd.edu/~mihir/ http://mathworld.wolfram.com/BirthdayAttack.html https://danielmiessler.com/study/birthday_attack/ https://www.sciencedirect.com/topics/computer-science/birthday-attack (a collection of links to other books and materials mentioning the birthday attack; some are relevant here, others less so) https://www.keylength.com/en/compare/ https://infoscience.epfl.ch/record/164526/files/NPDF-22.pdf (Lenstra) https://mailarchive.ietf.org/arch/msg/ipsec/T1woQuwh1Ccoz6fWWFDBETBllaY (brief but good discussion of rekeying lifetime) https://en.wikipedia.org/wiki/Key_size (not very well written) https://www.keylength.com/en/3/ (includes relative strength of algos) https://blog.cloudflare.com/why-are-some-keys-small/ (good article) https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf (2018 recommendations, 574 references) https://link.springer.com/chapter/10.1007/3-540-44448-3_42 (Abdalla & Bellare)

https://eprint.iacr.org/2012/623 Cryptology eprint by David McGrew, discussed in messages such as https://mailarchive.ietf.org/arch/msg/ipsec/7V6ry4le7eU3v283uF5D3Y9lCsU and https://mailarchive.ietf.org/arch/msg/ipsec/fQciMNrxTdsM3CNTAP0F4CKjjtc The former of those also discusses biclique attacks on AES.

8.5 Crypto law:

http://www.cryptolaw.org/cls2.htm (country-by-country crypto export status as of 2013) https://en.wikipedia.org/wiki/40-bit_encryption (the U.S. had a 40-bit limit in 1990s; not clear to me when it was relaxed)

8.6 Bitcoin:

https://driveinsider.com/the-quantum-attack-on-bitcoin/ https://www.bitcoinmarketjournal.com/how-many-people-use-bitcoin/ https://arxiv.org/abs/1710.10377 https://cointelegraph.com/news/quantum-computing-vs-blockchain-impact-on-cryptog (not very accurate)

8.7 Others:

https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5208 (38th FOCS) https://cloud.google.com/security/encryption-at-rest/default-encryption/ (describes some of Google's current practices and planned upgrades wrt encryption) https://info.townsendsecurity.com/km-trends https://en.wikipedia.org/wiki/Key_management https://www.theregister.co.uk/2019/06/26/amd_epyc_key_security_flaw/

U.S. presidential candidate Andrew Yang has a policy on quantum computing and encryption standards! https://www.yang2020.com/policies/quantum-computing/

https://blogs.technet.microsoft.com/secguide/2014/04/07/why-were-not-recommending-fips-mode-anymore/ https://en.wikipedia.org/wiki/FIPS_140-2

Mozilla moved to limit the lifetime of TLS connections in 2016: https://bugzilla.mozilla.org/show_bug.cgi?id=1268745 (found via SWEET32)

This was based in part on a paper by Luykx and Paterson, apparently, though the PDF is dated later than the Mozilla web page above. Atul Luykx and Kenneth G. Paterson Limits on Authenticated Encryption Use in TLS

Weak PRNGs, including Microsoft's, and RSA/NSA elliptic curve generator: https://en.wikipedia.org/wiki/Random_number_generator_attack

Appendix A: Bicoin and Quantum Computers

⇒someone needs to take this one on, too...

I promised those of you who stuck around some things on bitcoin and quantum. Here are just a few random links, not stuff I have studied seriously. There has been a bunch in the news lately, and I don't have time right now to sort out what's what.

Quantum attacks on Bitcoin: https://driveinsider.com/the-quantum-attack-on-bitcoin/

https://www.bitcoinmarketjournal.com/how-many-people-use-bitcoin/ found via https://twitter.com/yayayday/status/1140214798367543297 says 32M Bitcoin wallets, 34% are active users, 7M active.

Marco Tomamichel with some data on level of vulnerability across the entire bitcoin universe: https://twitter.com/marcotomamichel/status/1138431744103686144 June 11, 2019 and follow-ups, including https://twitter.com/marcotomamichel/status/1138438446622470144

Appendix B: Simple Coding Examples

⇒Get cocori to contribute here.

1. Entropy

From the fantastic website RosettaCode.org[64], we have a simple example in Python of calculating the entropy in bits per byte:

```
import math
from collections import Counter

def entropy(s):
    p, lns = Counter(s), float(len(s))
    return -sum( count/lns * math.log(count/lns, 2) for count in p.values())

entropy("1223334444")
```

(This function can actually calculate entropy of strings of other types besides characters, as well.)

2. Birthday Paradox/Collision Probability

Some ad hoc Mathematica code for dinking around with some of the numbers:

```
NoCollisionProbability[n_, D_] := N[Factorial[D]/Factorial[D - n]/D^n]

NoCollisionProbability[23, 365]
0.492703

NoCollisionProbability[2^10, 2^20]
0.606728

Stirling[n_] := Sqrt[2*\[Pi]*n]*(n/E)^n

NoCollisionProbabilityApprox[n_, D_] :=
 N[Stirling[D]/Stirling[D - n]/D^n]

NoCollisionProbabilityApprox[2^10, 2^20]
0.6067282267199

NoCollisionProbabilityApprox[2^12, 2^24]
0.606580027339
```

(That last example took several minutes on my laptop. Be very careful running the above code with larger numbers, it's very easy to start a computation that won't complete in your lifetime, and Mathematica isn't as friendly about stopping computations in a reasonable state as one might like.)

A good analysis of the birthday bound appears in Sec. 2.2 of the SWEET32 paper by Bhargavan and Leurent.

The square root limit as n and d grow large, in Mathematica:

```
N[Sqrt[E]]
1.64827

N[1 - 1/Sqrt[E]]
0.393469
```

3. QKD

BB84 in Qiskit: `https://github.com/qiskit-community/may4_challenge_exercises/blob/master/ex03/Challenge3_BB84_Solutions.ipynb`

Appendix C: Performance Testing

The data in Tab. I was taken using the `openssl speed` command.

```
VanMetedneysMBP:what-crypto rdv$ openssl speed
Doing mdc2 for 3s on 16 size blocks: 3269132 mdc2's in 2.99s
Doing mdc2 for 3s on 64 size blocks: 876412 mdc2's in 3.00s
Doing mdc2 for 3s on 256 size blocks: 224903 mdc2's in 3.00s
Doing mdc2 for 3s on 1024 size blocks: 56808 mdc2's in 3.00s
Doing mdc2 for 3s on 8192 size blocks: 7145 mdc2's in 2.99s
Doing md4 for 3s on 16 size blocks: 15913709 md4's in 3.00s
Doing md4 for 3s on 64 size blocks: 10944053 md4's in 2.99s
Doing md4 for 3s on 256 size blocks: 6174163 md4's in 2.99s
Doing md4 for 3s on 1024 size blocks: 2243828 md4's in 3.00s
Doing md4 for 3s on 8192 size blocks: 320567 md4's in 3.00s
Doing md5 for 3s on 16 size blocks: 13281992 md5's in 3.00s
Doing md5 for 3s on 64 size blocks: 10051176 md5's in 3.00s
Doing md5 for 3s on 256 size blocks: 5492455 md5's in 3.00s
Doing md5 for 3s on 1024 size blocks: 1914918 md5's in 3.00s
Doing md5 for 3s on 8192 size blocks: 282452 md5's in 2.99s
Doing hmac(md5) for 3s on 16 size blocks: 12379214 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 64 size blocks: 9300850 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 256 size blocks: 5332434 hmac(md5)'s in 3.01s
Doing hmac(md5) for 3s on 1024 size blocks: 1903841 hmac(md5)'s in 2.99s
Doing hmac(md5) for 3s on 8192 size blocks: 285053 hmac(md5)'s in 3.00s
Doing sha1 for 3s on 16 size blocks: 15194088 sha1's in 3.00s
Doing sha1 for 3s on 64 size blocks: 11467843 sha1's in 3.00s
Doing sha1 for 3s on 256 size blocks: 6788008 sha1's in 2.99s
Doing sha1 for 3s on 1024 size blocks: 2652441 sha1's in 3.00s
Doing sha1 for 3s on 8192 size blocks: 395370 sha1's in 3.00s
Doing sha256 for 3s on 16 size blocks: 17701259 sha256's in 3.00s
Doing sha256 for 3s on 64 size blocks: 9589067 sha256's in 2.99s
Doing sha256 for 3s on 256 size blocks: 4492836 sha256's in 3.00s
Doing sha256 for 3s on 1024 size blocks: 1372811 sha256's in 2.99s
Doing sha256 for 3s on 8192 size blocks: 185328 sha256's in 3.00s
Doing sha512 for 3s on 16 size blocks: 11992919 sha512's in 3.00s
Doing sha512 for 3s on 64 size blocks: 11759327 sha512's in 3.00s
Doing sha512 for 3s on 256 size blocks: 4895150 sha512's in 2.99s
Doing sha512 for 3s on 1024 size blocks: 1797904 sha512's in 3.00s
Doing sha512 for 3s on 8192 size blocks: 267826 sha512's in 3.00s
Doing whirlpool for 3s on 16 size blocks: 8194016 whirlpool's in 3.00s
Doing whirlpool for 3s on 64 size blocks: 4385938 whirlpool's in 2.99s
Doing whirlpool for 3s on 256 size blocks: 1746529 whirlpool's in 2.99s
Doing whirlpool for 3s on 1024 size blocks: 530664 whirlpool's in 3.00s
Doing whirlpool for 3s on 8192 size blocks: 71206 whirlpool's in 3.00s
Doing rmd160 for 3s on 16 size blocks: 7831461 rmd160's in 3.00s
Doing rmd160 for 3s on 64 size blocks: 4582251 rmd160's in 2.99s
Doing rmd160 for 3s on 256 size blocks: 2085955 rmd160's in 3.00s
Doing rmd160 for 3s on 1024 size blocks: 658382 rmd160's in 3.00s
Doing rmd160 for 3s on 8192 size blocks: 89699 rmd160's in 3.00s
Doing rc4 for 3s on 16 size blocks: 147178841 rc4's in 3.00s
Doing rc4 for 3s on 64 size blocks: 39947185 rc4's in 3.00s
Doing rc4 for 3s on 256 size blocks: 10038183 rc4's in 3.00s
Doing rc4 for 3s on 1024 size blocks: 2541336 rc4's in 3.00s
Doing rc4 for 3s on 8192 size blocks: 318930 rc4's in 3.00s
Doing des cbc for 3s on 16 size blocks: 16737820 des cbc's in 3.00s
Doing des cbc for 3s on 64 size blocks: 4263544 des cbc's in 2.99s
Doing des cbc for 3s on 256 size blocks: 1079074 des cbc's in 3.00s
Doing des cbc for 3s on 1024 size blocks: 270286 des cbc's in 3.00s
Doing des cbc for 3s on 8192 size blocks: 33767 des cbc's in 3.00s
Doing des ede3 for 3s on 16 size blocks: 6480362 des ede3's in 3.00s
Doing des ede3 for 3s on 64 size blocks: 1647195 des ede3's in 3.00s
Doing des ede3 for 3s on 256 size blocks: 410494 des ede3's in 3.00s
Doing des ede3 for 3s on 1024 size blocks: 102971 des ede3's in 3.00s
Doing des ede3 for 3s on 8192 size blocks: 12733 des ede3's in 3.00s
Doing aes-128 cbc for 3s on 16 size blocks: 29392770 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 64 size blocks: 8005053 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 256 size blocks: 2022753 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 1024 size blocks: 521883 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 8192 size blocks: 65725 aes-128 cbc's in 3.00s
Doing aes-192 cbc for 3s on 16 size blocks: 24767679 aes-192 cbc's in 3.00s
Doing aes-192 cbc for 3s on 64 size blocks: 6660982 aes-192 cbc's in 3.00s
Doing aes-192 cbc for 3s on 256 size blocks: 1717962 aes-192 cbc's in 3.00s
Doing aes-192 cbc for 3s on 1024 size blocks: 437476 aes-192 cbc's in 3.00s
Doing aes-192 cbc for 3s on 8192 size blocks: 54041 aes-192 cbc's in 3.00s
Doing aes-256 cbc for 3s on 16 size blocks: 21373898 aes-256 cbc's in 3.00s
Doing aes-256 cbc for 3s on 64 size blocks: 5798699 aes-256 cbc's in 3.00s
Doing aes-256 cbc for 3s on 256 size blocks: 1457112 aes-256 cbc's in 3.00s
Doing aes-256 cbc for 3s on 1024 size blocks: 358944 aes-256 cbc's in 3.00s
Doing aes-256 cbc for 3s on 8192 size blocks: 46453 aes-256 cbc's in 3.00s
Doing aes-128 ige for 3s on 16 size blocks: 29310580 aes-128 ige's in 3.00s
Doing aes-128 ige for 3s on 64 size blocks: 7706103 aes-128 ige's in 3.00s
Doing aes-128 ige for 3s on 256 size blocks: 1912179 aes-128 ige's in 3.00s
Doing aes-128 ige for 3s on 1024 size blocks: 484092 aes-128 ige's in 3.00s
Doing aes-128 ige for 3s on 8192 size blocks: 60676 aes-128 ige's in 3.00s
Doing aes-192 ige for 3s on 16 size blocks: 25157305 aes-192 ige's in 3.00s
Doing aes-192 ige for 3s on 64 size blocks: 6389211 aes-192 ige's in 3.00s
Doing aes-192 ige for 3s on 256 size blocks: 1612926 aes-192 ige's in 3.00s
Doing aes-192 ige for 3s on 1024 size blocks: 407802 aes-192 ige's in 3.00s
Doing aes-192 ige for 3s on 8192 size blocks: 51005 aes-192 ige's in 3.00s
Doing aes-256 ige for 3s on 16 size blocks: 21622758 aes-256 ige's in 3.00s
Doing aes-256 ige for 3s on 64 size blocks: 5567261 aes-256 ige's in 2.99s
Doing aes-256 ige for 3s on 256 size blocks: 1405116 aes-256 ige's in 3.01s
Doing aes-256 ige for 3s on 1024 size blocks: 346643 aes-256 ige's in 2.99s
Doing aes-256 ige for 3s on 8192 size blocks: 42047 aes-256 ige's in 2.99s
Doing ghash for 3s on 16 size blocks: 293141598 ghash's in 3.00s
Doing ghash for 3s on 64 size blocks: 275376555 ghash's in 2.99s
Doing ghash for 3s on 256 size blocks: 107889947 ghash's in 3.00s
Doing ghash for 3s on 1024 size blocks: 29614410 ghash's in 2.98s
Doing ghash for 3s on 8192 size blocks: 3639297 ghash's in 2.99s
Doing camellia-128 cbc for 3s on 16 size blocks: 22221376 camellia-128 cbc's in 3.01s
Doing camellia-128 cbc for 3s on 64 size blocks: 9161466 camellia-128 cbc's in 3.00s
Doing camellia-128 cbc for 3s on 256 size blocks: 2581860 camellia-128 cbc's in 2.99s
Doing camellia-128 cbc for 3s on 1024 size blocks: 681034 camellia-128 cbc's in 3.00s
Doing camellia-128 cbc for 3s on 8192 size blocks: 84919 camellia-128 cbc's in 3.00s
Doing camellia-192 cbc for 3s on 16 size blocks: 21323655 camellia-192 cbc's in 3.00s
Doing camellia-192 cbc for 3s on 64 size blocks: 7070729 camellia-192 cbc's in 3.00s
Doing camellia-192 cbc for 3s on 256 size blocks: 1898937 camellia-192 cbc's in 2.99s
Doing camellia-192 cbc for 3s on 1024 size blocks: 496149 camellia-192 cbc's in 3.00s
Doing camellia-192 cbc for 3s on 8192 size blocks: 63676 camellia-192 cbc's in 2.99s
Doing camellia-256 cbc for 3s on 16 size blocks: 20392955 camellia-256 cbc's in 3.00s
Doing camellia-256 cbc for 3s on 64 size blocks: 6837414 camellia-256 cbc's in 2.99s
Doing camellia-256 cbc for 3s on 256 size blocks: ^@1897179 camellia-256 cbc's in 2.99s
Doing camellia-256 cbc for 3s on 1024 size blocks: 492045 camellia-256 cbc's in 2.98s
Doing camellia-256 cbc for 3s on 8192 size blocks: 58736 camellia-256 cbc's in 2.98s
Doing idea cbc for 3s on 16 size blocks: 17488707 idea cbc's in 3.00s
Doing idea cbc for 3s on 64 size blocks: 4025928 idea cbc's in 2.96s
Doing idea cbc for 3s on 256 size blocks: 1112768 idea cbc's in 2.98s
Doing idea cbc for 3s on 1024 size blocks: 286779 idea cbc's in 2.99s
Doing idea cbc for 3s on 8192 size blocks: 35635 idea cbc's in 2.99s
Doing seed cbc for 3s on 16 size blocks: 17660765 seed cbc’s in 2.98s
Doing seed cbc for 3s on 64 size blocks: 4533929 seed cbc’s in 3.00s
Doing seed cbc for 3s on 256 size blocks: 1134251 seed cbc’s in 2.99s
Doing seed cbc for 3s on 1024 size blocks: 280526 seed cbc’s in 2.99s
Doing seed cbc for 3s on 8192 size blocks: 35611 seed cbc’s in 3.00s
Doing rc2 cbc for 3s on 16 size blocks: 10085165 rc2 cbc’s in 2.99s
Doing rc2 cbc for 3s on 64 size blocks: 2469228 rc2 cbc’s in 2.98s
Doing rc2 cbc for 3s on 256 size blocks: 624907 rc2 cbc’s in 2.98s
Doing rc2 cbc for 3s on 1024 size blocks: 162685 rc2 cbc’s in 2.99s
Doing rc2 cbc for 3s on 8192 size blocks: 20256 rc2 cbc’s in 3.00s
Doing blowfish cbc for 3s on 16 size blocks: 24689244 blowfish cbc’s in 2.99s
Doing blowfish cbc for 3s on 64 size blocks: 6359842 blowfish cbc’s in 3.00s
Doing blowfish cbc for 3s on 256 size blocks: 1679425 blowfish cbc’s in 3.00s
Doing blowfish cbc for 3s on 1024 size blocks: 419156 blowfish cbc’s in 3.00s
Doing blowfish cbc for 3s on 8192 size blocks: 50507 blowfish cbc’s in 2.99s
Doing cast cbc for 3s on 16 size blocks: 25855056 cast cbc’s in 3.00s
Doing cast cbc for 3s on 64 size blocks: 6664686 cast cbc’s in 2.99s
Doing cast cbc for 3s on 256 size blocks: 1629534 cast cbc’s in 3.00s
Doing cast cbc for 3s on 1024 size blocks: 418633 cast cbc’s in 3.00s
Doing cast cbc for 3s on 8192 size blocks: 52421 cast cbc’s in 2.99s
Doing 512 bit private rsa’s for 10s: 226983 512 bit private RSA’s in 9.99s
Doing 512 bit public rsa’s for 10s: 2755836 512 bit public RSA’s in 9.93s
Doing 1024 bit private rsa’s for 10s: 112814 1024 bit private RSA’s in 9.99s
Doing 1024 bit public rsa’s for 10s: 1460729 1024 bit public RSA’s in 9.97s
Doing 2048 bit private rsa’s for 10s: 17128 2048 bit private RSA’s in 9.96s
Doing 2048 bit public rsa’s for 10s: 573071 2048 bit public RSA’s in 9.98s
Doing 4096 bit private rsa’s for 10s: 2738 4096 bit private RSA’s in 9.98s
Doing 4096 bit public rsa’s for 10s: 171360 4096 bit public RSA’s in 9.96s
Doing 512 bit sign dsa’s for 10s: 219734 512 bit DSA signs in 9.98s
Doing 512 bit verify dsa’s for 10s: 292994 512 bit DSA verify in 9.98s
Doing 1024 bit sign dsa’s for 10s: 119931 1024 bit DSA signs in 9.98s
Doing 1024 bit verify dsa’s for 10s: 138893 1024 bit DSA verify in 9.97s
Doing 2048 bit sign dsa’s for 10s: 45063 2048 bit DSA signs in 9.98s
Doing 2048 bit verify dsa’s for 10s: 50192 2048 bit DSA verify in 9.98s
Doing 160 bit sign ecdsa’s for 10s: 175494 160 bit ECDSA signs in 9.97s
Doing 160 bit verify ecdsa’s for 10s: 54783 160 bit ECDSA verify in 9.98s
Doing 192 bit sign ecdsa’s for 10s: 162277 192 bit ECDSA signs in 9.99s
Doing 192 bit verify ecdsa’s for 10s: 45273 192 bit ECDSA verify in 9.97s
Doing 224 bit sign ecdsa’s for 10s: 117692 224 bit ECDSA signs in 9.97s
Doing 224 bit verify ecdsa’s for 10s: 34452 224 bit ECDSA verify in 9.98s
Doing 256 bit sign ecdsa’s for 10s: 276147 256 bit ECDSA signs in 9.98s
Doing 256 bit verify ecdsa’s for 10s: 138536 256 bit ECDSA verify in 9.97s
Doing 384 bit sign ecdsa’s for 10s: 57823 384 bit ECDSA signs in 9.98s
Doing 384 bit verify ecdsa’s for 10s: 14624 384 bit ECDSA verify in 9.99s
Doing 521 bit sign ecdsa’s for 10s: 28064 521 bit ECDSA signs in 9.98s
Doing 521 bit verify ecdsa’s for 10s: 7125 521 bit ECDSA verify in 9.99s
Doing 163 bit sign ecdsa’s for 10s: 62212 163 bit ECDSA signs in 9.99s
Doing 163 bit verify ecdsa’s for 10s: 29351 163 bit ECDSA verify in 9.98s
Doing 233 bit sign ecdsa’s for 10s: 32219 233 bit ECDSA signs in 9.99s
Doing 233 bit verify ecdsa’s for 10s: 22655 233 bit ECDSA verify in 9.99s
Doing 283 bit sign ecdsa’s for 10s: 22249 283 bit ECDSA signs in 9.99s
Doing 283 bit verify ecdsa’s for 10s: 13129 283 bit ECDSA verify in 9.99s
Doing 409 bit sign ecdsa’s for 10s: 9775 409 bit ECDSA signs in 9.98s
Doing 409 bit verify ecdsa’s for 10s: 8376 409 bit ECDSA verify in 9.99s
Doing 571 bit sign ecdsa’s for 10s: 4856 571 bit ECDSA signs in 9.99s
Doing 571 bit verify ecdsa’s for 10s: 3653 571 bit ECDSA verify in 9.98s
Doing 163 bit sign ecdsa’s for 10s: 61897 163 bit ECDSA signs in 9.98s
Doing 163 bit verify ecdsa’s for 10s: 28038 163 bit ECDSA verify in 9.99s
Doing 233 bit sign ecdsa’s for 10s: 32212 233 bit ECDSA signs in 9.99s
Doing 233 bit verify ecdsa’s for 10s: 21874 233 bit ECDSA verify in 9.99s
Doing 283 bit sign ecdsa’s for 10s: 22119 283 bit ECDSA signs in 9.99s
Doing 283 bit verify ecdsa’s for 10s: 12022 283 bit ECDSA verify in 9.97s
Doing 409 bit sign ecdsa’s for 10s: 9851 409 bit ECDSA signs in 9.98s
Doing 409 bit verify ecdsa’s for 10s: 7826 409 bit ECDSA verify in 9.99s
Doing 571 bit sign ecdsa’s for 10s: 4730 571 bit ECDSA signs in 9.98s
Doing 571 bit verify ecdsa’s for 10s: 3394 571 bit ECDSA verify in 9.99s
Doing 160 bit  ecdh’s for 10s: 65467 160-bit ECDH ops in 9.97s
Doing 192 bit  ecdh’s for 10s: 53417 192-bit ECDH ops in 9.97s
Doing 224 bit  ecdh’s for 10s: 40923 224-bit ECDH ops in 9.98s
Doing 256 bit  ecdh’s for 10s: 213202 256-bit ECDH ops in 9.98s
Doing 384 bit  ecdh’s for 10s: 17258 384-bit ECDH ops in 9.99s
Doing 521 bit  ecdh’s for 10s: 8544 521-bit ECDH ops in 9.98s
Doing 163 bit  ecdh’s for 10s: 59866 163-bit ECDH ops in 9.99s
Doing 233 bit  ecdh’s for 10s: 47039 233-bit ECDH ops in 9.98s
Doing 283 bit  ecdh’s for 10s: 26741 283-bit ECDH ops in 9.97s
Doing 409 bit  ecdh’s for 10s: 17323 409-bit ECDH ops in 9.98s
Doing 571 bit  ecdh’s for 10s: 7582 571-bit ECDH ops in 9.99s
Doing 163 bit  ecdh’s for 10s: 56840 163-bit ECDH ops in 9.98s
Doing 233 bit  ecdh’s for 10s: 45098 233-bit ECDH ops in 9.99s
Doing 283 bit  ecdh’s for 10s: 25454 283-bit ECDH ops in 9.99s
Doing 409 bit  ecdh’s for 10s: 16178 409-bit ECDH ops in 9.98s
Doing 571 bit  ecdh’s for 10s: 7069 571-bit ECDH ops in 9.99s
OpenSSL 1.0.2o  27 Mar 2018
built on: reproducible build, date unspecified
options:bn(64,64) rc4(ptr,int) des(idx,cisc,16,int) aes(partial) idea(int) blowfish(idx)
compiler: x86_64-apple-darwin13.4.0-clang -D_FORTIFY_SOURCE=2 -mmacosx-version-min=10.9 -march
The ’numbers’ are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
md2                  0.00         0.00         0.00         0.00         0.00
mdc2             17493.68k    18696.79k    19191.72k    19390.46k    19575.87k
md4              84873.11k   234253.98k   528623.99k   765893.29k   875361.62k
md5              70837.29k   214425.09k   468689.49k   653625.34k   773861.80k
hmac(md5)        66022.47k   198418.13k   453522.63k   652017.79k   778384.73k
sha1             81035.14k   244647.32k   581180.62k   905366.53k  1079623.68k
rmd160           41767.79k    98081.63k   178001.49k   224727.72k   244938.07k
rc4             784953.82k   852206.61k   856591.62k   867442.69k   870891.52k
des cbc          89268.37k    91259.80k    92080.98k    92257.62k    92206.42k
des ede3         34561.93k    35140.16k    35028.82k    35147.43k    34769.58k
idea cbc         93273.10k    87047.09k    95593.49k    98214.61k    97632.75k
seed cbc         94822.90k    96723.82k    97113.13k    96073.12k    97241.77k
rc2 cbc          53967.44k    53030.40k    53683.29k    55715.53k    55312.38k
rc5-32/12 cbc        0.00         0.00         0.00         0.00         0.00
blowfish cbc    132116.36k   135676.63k   143310.93k   143071.91k   138379.04k
cast cbc        137893.63k   142655.49k   139053.57k   142893.40k   143623.02k
aes-128 cbc     156761.44k   171345.62k   172608.26k   178136.06k   179473.07k
aes-192 cbc     132094.29k   142100.95k   146599.42k   149325.14k   147567.96k
aes-256 cbc     113994.12k   123705.58k   124340.22k   122519.55k   126847.66k
camellia-128 cbc   118120.27k   195444.61k   221055.57k   232459.61k   231885.48k
camellia-192 cbc   113726.16k   150842.22k   162584.57k   169352.19k   174459.46k
camellia-256 cbc   108762.43k   146352.67k   162434.05k   169078.55k   161464.87k
sha256           94406.71k   205250.93k   383388.67k   470153.33k   506068.99k
sha512           63962.23k   250865.64k   419116.52k   613684.57k   731343.53k
whirlpool        43701.42k    93879.61k   149535.59k   181133.31k   194439.85k
aes-128 ige     156323.09k   164396.86k   163172.61k   165236.74k   165685.93k
aes-192 ige     134172.29k   136303.17k   137636.35k   139196.42k   139277.65k
aes-256 ige     115321.38k   119165.45k   119504.88k   118716.53k   115200.34k
```

```
ghash          1563421.86k  5894347.67k  9206608.81k 10176226.79k  9970943.49k
                  sign    verify    sign/s verify/s
rsa  512 bits 0.000044s 0.000004s  22721.0 277526.3
rsa 1024 bits 0.000089s 0.000007s  11292.7 146512.4
rsa 2048 bits 0.000582s 0.000017s   1719.7  57421.9
rsa 4096 bits 0.003645s 0.000058s    274.3  17204.8
                  sign    verify    sign/s verify/s
dsa  512 bits 0.000045s 0.000034s  22017.4  29358.1
dsa 1024 bits 0.000083s 0.000072s  12017.1  13931.1
dsa 2048 bits 0.000221s 0.000199s   4515.3   5029.3
                              sign    verify    sign/s verify/s
 160 bit ecdsa (secp160r1)   0.0001s   0.0002s  17602.2   5489.3
 192 bit ecdsa (nistp192)   0.0001s   0.0002s  16243.9   4540.9
 224 bit ecdsa (nistp224)   0.0001s   0.0003s  11804.6   3452.1
 256 bit ecdsa (nistp256)   0.0000s   0.0001s  27670.0  13895.3
 384 bit ecdsa (nistp384)   0.0002s   0.0007s   5793.9   1463.9
 521 bit ecdsa (nistp521)   0.0004s   0.0014s   2812.0    713.2
 163 bit ecdsa (nistk163)   0.0002s   0.0003s   6227.4   2941.0
 233 bit ecdsa (nistk233)   0.0003s   0.0004s   3225.1   2267.8
 283 bit ecdsa (nistk283)   0.0004s   0.0008s   2227.1   1314.2
 409 bit ecdsa (nistk409)   0.0010s   0.0012s    979.5    838.4
 571 bit ecdsa (nistk571)   0.0021s   0.0027s    486.1    366.0
 163 bit ecdsa (nistb163)   0.0002s   0.0004s   6202.1   2806.6
 233 bit ecdsa (nistb233)   0.0003s   0.0005s   3224.4   2189.6
 283 bit ecdsa (nistb283)   0.0005s   0.0008s   2214.1   1205.8
 409 bit ecdsa (nistb409)   0.0010s   0.0013s    987.1    783.4
 571 bit ecdsa (nistb571)   0.0021s   0.0029s    473.9    339.7
                              op      op/s
 160 bit ecdh (secp160r1)   0.0002s   6566.4
 192 bit ecdh (nistp192)   0.0002s   5357.8
 224 bit ecdh (nistp224)   0.0002s   4100.5
 256 bit ecdh (nistp256)   0.0000s  21362.9
 384 bit ecdh (nistp384)   0.0006s   1727.5
 521 bit ecdh (nistp521)   0.0012s    856.1
 163 bit ecdh (nistk163)   0.0002s   5992.6
 233 bit ecdh (nistk233)   0.0002s   4713.3
 283 bit ecdh (nistk283)   0.0004s   2682.1
 409 bit ecdh (nistk409)   0.0006s   1735.8
 571 bit ecdh (nistk571)   0.0013s    759.0
 163 bit ecdh (nistb163)   0.0002s   5695.4
 233 bit ecdh (nistb233)   0.0002s   4514.3
 283 bit ecdh (nistb283)   0.0004s   2547.9
 409 bit ecdh (nistb409)   0.0006s   1621.0
```

Appendix D: Unsorted Notes

https://brunorijsman.github.io/openssl-qkd/

quantum random number generators:

standalone: Tamura & Shikano on https://arxiv.org/abs/1906.04410

Bell: Shen ... Nam, Scarani, Kuritseifer https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.121.150402 https://arxiv.org/abs/1805.02828

practical device-independent qcrypt via entropy Arnon-Friedman, ..., Renner, Vidick https://www.nature.com/articles/s41467-017-02307-4

WIDE PKI certificate: SHA-256 with RSA Encryption (1.2.840.113549.1.1.11) Exponent 65537 key size 2048 bits

(SHA-256 is one type of SHA-2; SHA-3 also exists? Why do we use 2?)

EV = Extended Validation Certificate gigazine.net/news/20190813-extended-validation-certificate/

SHA-1 broken: https://sha-mbles.github.io/ https://link.springer.com/chapter/10.1007%2F978-3-030-17659-4_18 https://eurocrypt.iacr.org/2019/program.html

cocori's post-quantum crypto Python. New version: https://colab.research.google.com/drive/1jnAY-Jopo8t5Cb0_sT2cNuMORFYnkfl2?usp=sharing Old version: https://colab.research.google.com/drive/1a4sh9PYCOyZO9dRsJpT17scLS3eMWU0T

1. A Twitter Exchange

```
sanketh
@__c1own
Replying to
@rdviii
Awesome series!! Quick comment on your statement in
(https://rdvlivefromtokyo.blogspot.com/2019/10/aes-advanced-encryption-standard.html)
that AES is easy to implement. It is not. See
https://cr.yp.to/antiforgery/cachetiming-20050414.pdf

https://twitter.com/__c1own/status/1216868406047191040
sanketh
@__c1own
Replying to
@__c1own
 and
@rdviii
Two quick comments on (https://rdvlivefromtokyo.blogspot.com/2019/11/21-playing-defense-211-en
1. Kolmogorov complexity is uncomputable (https://web.archive.org/web/20190326105001/ http://w
2. You hint at this in the post, but to be explicit, in crypto, we
care about min-entropy and not entropy.
```

https://cr.yp.to/antiforgery/cachetiming-20050414.pdf

A nice blog article by @__c1own on hybrid quantum-classical computation, in the context of attacking crypto. https://c1own.com/blog/2019/two-open-problems-hybrid-quantum-attacks-on-crypto/

ACK Sanketh (c1own)

https://cr.yp.to/snuffle/design.pdf

possible (elliptic?) curve vulnerability: https://news.ycombinator.com/item?id=22048619&fbclid=IwAR0HTje3SlkFTaryK76gPDn_ss1k6n-yAGZlVF8bN6xvqLNBuZ33c8XTnnc

Grassl et al. on AES w/ Grover: https://link.springer.com/chapter/10.1007/978-3-319-29360-8_3 AES https://eprint.iacr.org/2019/272.pdf new AES paper (with short but relevant refs): https://ieeexplore.ieee.org/document/8961201

New factoring record: https://www.johndcook.com/blog/2019/12/03/new-rsa-factoring/ https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2019-December/001139.html https://en.wikipedia.org/wiki/RSA_Factoring_Challenge

https://arxiv.org/pdf/1804.00200.pdf

More M$ on elliptic: https://eprint.iacr.org/2017/598.pdf

87% of web traffic was encrypted, as of Jan. 2019: https://duo.com/decipher/encryption-privacy-in-the-internet-trends-report

and by October of that year it was 90%: https://meterpreter.org/https-encryption-traffic/

Both of those seem to point back to Fortinet reports, but don't link to the exact one. Best I can find is 3Q18: https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/threat-report-q3-2018.pdf which says hovered around 50% throughout 2016, then rose sharply to 72% by 3Q18.

Apparently *old* discussion of short keys in SSL: http://www.geocities.ws/rahuljg/Attacks_on_SSL.htm

Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) is a security exploit against HTTPS when using HTTP compression. BREACH is built based on the CRIME security exploit. https://en.wikipedia.org/wiki/BREACH

CRIME is maybe closer to what I'm looking for: https://en.wikipedia.org/wiki/CRIME

Refers to (original?) paper on leakage of info via compression-over-encryption: https://link.springer.com/chapter/10.1007%2F3-540-45661-9_21

2. From ddp

just fast keying: http://www1.cs.columbia.edu/~angelos/Papers/jfk-tissec.pdf

IKEv1 was a group compromise, almost no one liked it. This was a valid criticism and probably should have been explored further. I note that many of the subsequent enhancements in IKEv2 involve ideas found here: http://www1.cs.columbia.edu/~angelos/Papers/jfk-tissec.pdf

But after three years of screaming at each other, it was time to ship something and see how it worked in the field. It's also a fair criticism that perfect is the enemy of ”it mostly works, and beats plaintext on the wire”.

”The keys used for the Child SAs, therefore, are the obvious target for traffic-based attacks, though the real prize is the keys for the IKE SA.”

The keys are equal, though you may think of the Phase 1 SA as being more important because the bulk keys are derived from it.

”I'm having a hard time imagining how to mount an effective attack against the IKE SA.”

good news!

Image of SKEYID: https://twitter.com/ddp/status/1263165272082403328

again, this is IKEv1

so the question you are posing is, what causes quantum keying material to wear out? It is a very good question.

Replying to @rdviii and @ddp People like to talk about D-H, RSA and Shor, because D-H and RSA are pretty easy to understand, but the actual use of the keys matters.

They both matter, but yes. The Phase 1 keys are used by IKE for connection setup and rekeying, traffic is encrypted under the bulk sesssion keys that must be periodically rekeyed, but that's dependent on the amount of traffic.

Depending on what algorithms you configure you may need to rekey more or less frequently, which is why it was left configurable on CryptoClusters. I'm purposefully and wantonly ignoring the Red Book which probably would point out that the entire stream is a covert channel.

Our TCSEC C2 and B1 evaluation stipulated that the network not be plugged in and DECwindows was disabled, there was a `SECURITY_POLICY` system parameter that was a bitmap of features to be disabled when running in the evaluated configuration.

3. More Unsorted Stuff

https://twitter.com/trimstray/status/1262985978618281986 Ethical hacking platforms/trainings/CTFs:

http://crackmes.one http://ringzer0ctf.com http://pwnable.kr http://ctflearn.com http://domgo.at http://pwnable.tw http://tryhackme.com http://ctfchallenge.co.uk http://cryptohack.org

—–

Dan Boneh's papers on SSL/TLS: http://crypto.stanford.edu/~dabo/pubs/pubsbytopic.html#S including some on quantum, going back as far as 95! Wow, and a paper on cracking DES on a DNA computer!

—–

Kenn White @kennwhite Replying to @rdviii and @danieljbaird I think @hanno or @FiloSottile might be able to point you in the right direction re a good corpus on attacks.

Daniel Baird @danieljbaird Replying to @danieljbaird

@JapanGraphica and @rdviii also @kennwhite

@matthew_d_green

@halvarflake

https://twitter.com/SteveBellovin/status/1256578124093022211 And then there was my mistake about sequence numbers in IPsec. @mattblaze wanted them; I said “no” and won. I then did some research that showed that he was right and I was wrong , so I led the effort to put them back in. See slide 43 of https://cs.columbia.edu/~smb/talks/why-ipsec.pdf.

NSA Military Cryptanalysis: https://www.nsa.gov/news-features/declassified-documents/military-cryptanalysis/ Friedman's guide to cryptanalysis from the war.

Replying to @rdviii i believe it would still be algorithm specific, so i'm not sure what can be drawn by looking at the specific cryptanalysis of des for example. in cases like these, dan and i always deferred to the cryptographers in the room; we had 3-4 participating and several from the nsa. 11:06 PM · May 22, 2020·Twitter Web App

Replying to @ddp and @rdviii again, we had the goal of algorithm flexibility

Searching publicly available NSA quantum documents: https://search.usa.gov/search?utf8=%E2%9C%93&affiliate=nsa_css&sort_by=&query=quantum

SHA-1 is a Shambles: https://eprint.iacr.org/2020/014.pdf https://sha-mbles.github.io/ https://www.openssh.com/txt/release-8.3

WeakDH is back online: https://weakdh.org/

Elliott et al., SIGCOMM 2003: https://arxiv.org/abs/quant-ph/0307049

—–

This might seem very far from linear algebra, but the terms ”linear” and ”affine” show up in linear cryptanalysis, with a special meaning derived from the same concepts we saw in class.

We saw a hint of the relationship between graph theory and linear algebra. In the Ritter literature survey, Buttyan and Vajda are cited as describing one important aspect of linear cryptanalysis as a graph problem that would take about a terabyte of memory. That was out of the question when they wrote their paper in 1995, but 1TB of RAM is no big deal today. I wonder if that is still a valid idea, and if anyone has followed up on it?

—

comments/observations from AQUAcamp

must understand layered communication protocols to follow this!

biclique is best attack against AES (but Aono-san says it's not practical)

Markus Grassl, Martin Roetteler on AES via Grover: https://arxiv.org/abs/1512.04965 requires tremendous resources. Only 3-7,000 logical qubits, but up to 2^{151} T gates?!?

Jogenfors (the QKD hacker?) on bitcoin: https://arxiv.org/abs/1604.01383

—–

https://news.yahoo.com/exclusive-russia-carried-out-a-stunning-breach-of-fbi-communications html?soc_src=community&soc_trk=tw

Darrell says IKEv1 is believed to be stronger post-quantum than IKEv2, but he hasn't told me why yet.

See Pirandola and Hoi-Kwong Lo for recent QKD reviews.

https://csrc.nist.gov/projects/post-quantum-cryptography

On Sept. 20, it was all over the news that Google is rumored to be claiming quantum supremacy: https://www.cnet.com/news/google-reportedly-attains-quantum-supremacy/

https://twitter.com/susurrusus/status/1142196496739291137?s=19 and substantial follow-on conversation, starting in part from some misguided post-quantum cryptocurrency: https://twitter.com/mochimocrypto/status/1175134812862058496 https://twitter.com/mochimocrypto/status/1175891543233708032 based on WOTS+. What are WOTS+ and XMSS? I assume DSA is digital signature algorithm. https://twitter.com/mochimocrypto/status/1175135848259624960 replied to by Biercuk https://twitter.com/MJBiercuk/status/1175394127841615872 https://twitter.com/jfitzsimons/status/1175919991322886145

https://blog.cloudflare.com/towards-post-quantum-cryptography-in-tls/amp/?__twitter_impression=true https://medium.com/altcoin-magazine/quantum-resistant-blockchain-and-cryptocurrency-the-full-analysis-in-seven-parts-part-6-7699 https://twitter.com/jtga_d/status/1175898478712635392

Might get either John Schanck or Douglas Stebila (recommended by Joe Fitzsimons) to review this. https://www.douglas.stebila.ca/ jschanck@uwaterloo.ca

New paper on AES and Grover https://arxiv.org/abs/1910.01700

Good info on status of DES/3-DES at https://en.wikipedia.org/wiki/Data_Encryption_Standard

Supplementary material listing all of the S-boxes, permutations and functions at https://en.wikipedia.org/wiki/DES_supplementary_material

Perlner survey of post-quantum crypto

pentesting (penetration testing)?

My QuantumInternet tweetstorm: https://twitter.com/rdviii/status/854443142304497665?s=19

PQC: https://www.scientificamerican.com/article/new-encryption-system-protects-data-from-qua

PUF (for making a key): https://en.wikipedia.org/wiki/Physical_unclonable_function

A quantum PUF: https://phys.org/news/2019-10-cryptography-secret-keys.html

https://www.schneier.com/blog/archives/2019/10/factoring_2048-.html

Shannon's a mathematical theory of cryptography https://www.iacr.org/museum/shannon45.html

Prof. Consheng DING's course on Cryptography and Security https://home.cse.ust.hk/faculty/cding/CSIT571/ slides on confusion & diffusion https://www.cse.ust.hk/faculty/cding/CSIT571/SLIDES/confdiffu.pdf

Shannon 1949 paper: Shannon, Claude. ”Communication Theory of Secrecy Systems”, Bell System Technical Journal, vol. 28(4), page 656–715, 1949. http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf

Scott Aaronson, certified RNG `https://arxiv.org/abs/1612.05903` also Mahadev, Vazirani, Vidick `https://arxiv.org/abs/1804.00640` `https://www.quantamagazine.org/how-to-turn-a-quantum-computer-into-the-ultimate-randomness-generator-20190619/`

More on PQC: `https://blog.cloudflare.com/the-tls-post-quantum-experiment/` `https://www.johndcook.com/blog/2019/10/23/quantum-supremacy-and-pqc/`

Program ”ent” on Linux distros?

IEICE special issue on crypto (2006, so it’s old): `https://d-nb.info/1170302556/34`

Kawamura on RNS Montgomery reduction: `https://d-nb.info/1170302556/34`

Corporate VPN clients (2018): `https://www.pcmag.com/picks/the-best-business-vpn-clients`

⇒NCP secure entry client for Win: IPsec tunnel, as well as “TCP encapsulation of IPSec with SSL headers” `https://www.pcmag.com/reviews/ncp-secure-entry-client-for-win3264`.

⇒OpenVPN: “all purpose” – what protocols? `https://www.pcmag.com/reviews/openvpn-243`.

⇒Greenbow IPsec VPN client: IPsec w/ a variety of keying methods; mentions SSL, but not sure how that works `https://www.pcmag.com/reviews/thegreenbow-ipsec-vpn-client`.

⇒Microsoft VPN client

⇒Woolf: Adding security decreases resilience.

⇒What’s a threat model, what does it mean, and who cares? Why does deploying this cost less than being threatened by this does?

⇒Woolf: Once again, the U.S. is trying to outlaw strong crypto. Bellovin, ISOC, Farber. We can’t have modern society without strong crypto.

`https://www.nict.go.jp/en/quantum/roadmap.html`

⇒There’s a great roadmap image somewhere; underneath `https://www.cryptrec.go.jp/index.html`?