# Cryptography

What do ancient Greeks from 1900 B.C. and blockchain have in common? Both have a keen interest in cryptography. The desire to send messages secretly across hostile environments, so that only the intended recipient could read and trust the message, has been around for some time. Today we understand cryptography to have four high-level goals.

* **Confidentiality.** The information must not be accessible to anyone who isn’t the intended audience of the message.
* **Integrity.** The information being transmitted must reach the recipient without alteration.
* **Non-repudiation.** The sender of the information is reliable and can be traced. The sender cannot deny sending the information.
* **Authentication.** The sender and receiver are confident of each other’s identity.

Sounds simple enough, right? It’s really interesting how cryptography has evolved through the ages. In this article, I’ll talk about an interesting historical context around cryptography and end up with how cryptography works in the modern world. I think you’ll be amazed at how far we’ve come, and yet we’re still solving 6000-year-old problems.

## Ancient Greece

A very long time ago, around 500 B.C., there was a Persian prince called Darius. He considered another gentleman, called Histiaeus, to be loyal and asked him to come to Susa as an advisor. Now, as it turned out, Histiaeus was quite unhappy in Susa and he felt he was almost a prisoner there. He wanted to escape and he had faith in his son-in-law Aristagoras to help him. Unfortunately, this was 500 B.C. and there was no Twitter or Facebook back then. Being able to securely communicate across a hostile environment was impossible. Or was it?

As it turns out, Histiaeus had a loyal slave. Histiaeus shaved the head of the slave and tattooed a message on his head, instructing Aristagoras to revolt against the Persians. Histiaeus then waited for the slave’s hair to grow back, and the slave then travelled to Aristagoras. The message was securely delivered.

What does all this have to do with encryption and cryptography? Histiaeus wanted to send an encrypted message to Aristagoras and he used a technique called steganography, which is the practice of concealing a message in another perfectly commonplace object. Histiaeus used his slave’s head, but you can use an image or a computer file to enclose your secure message.

You may think this is far-fetched, but it isn’t. Have you ever written a message in an invisible ink?

In World War II, Ms. Velvalee Dickinson, a spy for Japan, was a dealer in dolls and frequently used them to send information about ship movements.

In Vietnam, prisoner of war Jeremiah Denton blinked his eyes in Morse code to spell out T-O-R-T-U-R-E.

Tesla once caught a leaker by sending identical emails with minor differences, creating a digital signature to identify someone who leaked information. Check out this more recent [tweet](%20tweet) (<https://twitter.com/elonmusk/status/1579101966453858305>) by Elon Musk, another version of the same catch.

“That is quite an interesting story. We sent what appeared to be identical emails to all, but each was actually coded with either one or two spaces between sentences, forming a binary signature that identified the leaker.”

Elon Musk

Now, as impressive as Histiaeus’s escapades were, let’s be honest. Finding a slave in today’s world and shaving his head is not only inefficient, but also probably illegal. Not to mention that it would make the slave quite mad.

## The Roman Empire

Sometimes I wonder what it must have been like to live in the Roman Empire. The Roman Empire stretched over many cultures, a vast geographic expanse. Much like today, in times of excess, not everyone was happy. Even back then, there were those trying to exchange messages who feared being caught. So, they invented something called the Caesar cipher. At a high level, it was quite simple. If I wanted to send you a message, for example, we’d agree ahead of time that I’d shift the letters of the alphabet by a fixed number. Let’s say we agree to shift everything by three. So, the alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ now becomes XYZABCDEFGHIJKLMNOPQRSTUVW. The message “THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG” now becomes “QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD”.

if someone was caught sending a secret message, the guards would think it was written in some foreign language. And they’d just let the person go. ORO. Can you decode that? You can use this tool (<https://www.boxentriq.com/code-breaking/caesar-cipher>) to help you out.

## Ancient Iraq

As smart as the Romans were, there was a geek in ancient Iraq in the 8th century A.D. called Al-Kindi. This guy was a mathematician. When all his friends went partying on Friday evening, he’d sit on his, well, I want to say computer, but you get the idea. He discovered an interesting thing called “frequency analysis.”

Frequency analysis is based on the fact that in any given stretch of written language, certain letters and combinations of letters occur with varying frequencies. Moreover, there is a characteristic distribution of letters that’s roughly the same for almost all samples of that language.

Did you know that the letter “e” is the most commonly used character in the English and Spanish languages? You can differentiate English and Spanish because the letter A occurs way more often in Spanish than in English. You can see the frequency analysis of English in **Figure 1** and the frequency analysis of Spanish in **Figure 2**.



Figure 1: English frequency analysis



Figure 2: Spanish frequency analysis

This created a real problem for the Caesar cipher. If I knew that the shady guy with the weird message was English, I could just examine his message and replace the most common alphabet with “E” and guess the shift in the cipher. In fact, with modern computers, I could do this super-fast for all known languages. Once patterns start to emerge, I could easily decipher the message.

Here’s a funny story. In World War II, Germans always ended their messages with Heil Hitler! This really helped the allies crack their secret messages. More on that shortly.

To get around this problem, a better version of Caesar cipher was created called the polyalphabetic cipher. The idea was that a secret word was exchanged ahead of time. The shift would be done per the letters of that secret word.

Let’s say the secret word was “sahil”, and the numbers this corresponds to are, 18 (S), 0 (A), 7 (H), 8 (I), and 11 (L). Now instead of shifting everything with “3” or a fixed number, we shift using this sequence of 18, 0, 7, 8, 1. So my message of “thisisfine” becomes LHPATKFPVP. The first letter (T) gets shifted by 18, the second letter (H) gets shifted by 0 and so on so forth. When you get to the sixth letter, where the secret word has run out of letters, you just repeat the order of the letters in the secret word. So the S of “is” shifts by 18, followed by 0 and so on. The same letter is going to represented in several different ways as you work through the message.

The clear advantage here is that the distribution of alphabets in frequency analysis becomes much flatter. In fact, for an infinite-length message with an infinite-length key, the frequency analysis is absolutely flat. In order to break this cipher, you’d need to guess the length of the key, and try to shift the length and match with the standard English frequency pattern.

That’s much harder, right? But it’s still not impossible for a modern computer to break.

This technique was further improved with a one-time pad. In this technique, plaintext is paired with a random, secret key (or pad). Then, each bit or character of the plaintext is encrypted by combining it with the corresponding bit or character from the pad. The key is as long as the text itself.

Let’s say the pad was five digits long. Using “Sahil,” the total number of possibilities here is 26\*26\*26\*26\*26 = 11881376. This is a huge number. How huge? A typical sheet of paper is 0.002” thick. A stack of 11881376 sheets would be 700 meters high (almost half a mile).

Did you know that in the 1960s, KGB agents used magnesium paper with one-time pads. This was a tiny notebook, as shown here (<http://www.ranum.com/security/computer_security/papers/otp-faq/>) that the Russians exchanged ahead of time. If someone was busted, they could just burn the notebook and it burned quickly and left no residue. Remember, indoor smoking was normal back then. German U-boats used a similar technique, where the ink dissolved in water. Or sometimes the pad was so small that they’d have a booklet of pages, and when they used a pad, they’d eat that sheet of paper.

So you see, if you could have a one-time pad exchanged ahead of time with a sufficiently large cipher, you could create perfect encryption. Encryption so good that even modern computers couldn’t crack it.

There‘s one problem though. How do you exchange that cipher or one time pad securely ahead of time? If that gets compromised, everything is compromised. I’ll get to that in a moment, but first let me tell you another interesting story from World War II.

## World War II and the Enigma Machine

World War II was awful. But it was also a triumph of human ingenuity, a story of good over evil. A lot of interesting inventions came out of the duress of need. Nazis took the polyalphabetic cipher approach to create a device called the Enigma machine. It was the size of an oversized briefcase and it needed a battery to run. It had three rotors inside it and what looked like a typewriter in front of it. The idea was that every time you pressed a key on the typewriter keypad area, a corresponding alphabet lit up that was your encoded version. In addition, every time you’d press the alphabets, the rotors moved, effectively changing the cipher. This meant that pressing E multiple times yielded a different outcome each time. This made cracking the cipher a lot harder because it was constantly changing.

There were, in fact, two versions of the Enigma machine. The civilian version that the banks used had those three rotors and that’s it. But the military version had a switchboard in front, which further added numerous permutations and combinations; the idea was that the sender and receiver exchanged patterns of switchboards ahead of time. These were printed sheets of paper that covered about the next month ahead, and they were exchanged before, say, a U boat went out into the ocean.

The sender and recipient had the same codes, and they ensured that they used the same pattern on the switchboard every day. They could safely communicate with each other, but as this pattern changed every day, it was impossible to crack. In fact, those codes were written using water-soluble ink, so even if you got hold of the machine, without the codes, it was useless.

Very ingenious. But the allies had an interesting trick up their sleeve. See, the Germans didn’t realize that they had unknowingly baked a weakness into the Enigma machine. Additionally, they weren’t following the best security practices.

The Germans had engineered a solution so that a letter never yielded the letter itself in the cipher version. This ended up being a weakness of the product. The Germans started the day with the weather report, titled, duly, “Wetterbericht.” And they would, of course, sign off with “Heil Hitler.” All the allies had to do was align the resulting alphabets with the cipher alphabets to know which letters from the unencrypted version matched which letters in the encrypted version. Then it was a matter of brute-forcing combinations using a huge noisy analog computer they had built to decipher the code. Additionally, over time, the Germans got lazy, and didn’t bother to change the code every day. It was just so much hassle really. And that made the allies’ job a lot easier.

What do we learn from this? First, don’t reinvent security. Modern algorithms are open but the keys are secure. Second. don’t re-engineer the algorithms, as they are time-tested. Third don’t get lazy and reuse the same password everywhere.

## A Common Problem

After World War II, the world changed rapidly. We already had algorithms that offered perfect security under perfect circumstances. As long as you could exchange codes ahead of time and followed all the rules, you could effectively create an encryption mechanism that was undefeatable. This worked fine for banks because you could exchange the keys securely ahead of time.

But what if you didn’t know which party you were communicating with ahead of time? For instance, when you open a browser, and visit https anything, how is the connection securely negotiated on the fly without you first having to visit the website and store a key there? Hang on. I’ve got another story to tell.

World War II ended with the hope of rebuilding a prosperous and peaceful future. But then the cold war started. Both the U.S. and Russia had thousands of missiles tipped with very powerful nuclear bombs. If an ICBM launches from Russia, it has approximately 13 minutes before it lands in the United States. Faster, if the missile is launched from a submarine. Today’s hypersonic missiles cut that time by a fifth or more.

A network of radar stations and satellites were built that had to coordinate automatically and securely to judge the trajectory of such a missile and even to confirm that such a missile was indeed an ICBM. There was no space for a mistake here, because the reply to such an attack would be the end of the planet. Conversely, when such satellites and radar stations communicate with each other, they need to have confidence that they are communicating securely, that their messages can’t be snooped or tampered with, and that indeed, whoever they are communicating with is who they think they are communicating with. You know, all the stuff cryptography deals with.

How do we securely exchange a key ahead of time among so many parties in the 13 minutes it takes for the missile to land in my backyard? We can’t.

We needed a better mechanism, something that lets us communicate securely, without having to exchange a key ahead of time.

## The Solution: Symmetric Key Exchange.

The solution lies in one-way functions. If I ask you what is 2 multiplied by 5 multiplied by 7 multiplied by 13, with some difficulty, you’d be able to tell me the answer is 910. If I asked you what prime numbers I need to multiply to get 912, it would probably take you longer. This is a great example that traversing from A->B is easier than B->A. This is an example of a one-way function. My example was quite simple, but some very smart scientists have come up with algorithms that are much better at this one-way problem. The best one-way problem I could come up with was climbing up a ladder. It‘s always easier to climb up rather than down, unless it’s involuntary.

Let’s take the problem we’re trying to solve. Joe and Jack wish to exchange a secret, but Joe and Jack have never met each other. Clara, in the middle, is a hacker who can hear every communication between Joe and Jack. How do Joe and Jack quickly and securely exchange a key in such a way that even if Clara listens to every message, Clara cannot get hold of that key, that shared secret Joe and Jack can use to communicate further.

For simplicity, let’s assume that the one way algorithm is colors. If I mix yellow and blue, I get green. But if I gave you a bucket of green, and asked you to separate it into yellow and blue, it’s a lot harder.

First, Joe and Jack agree on a color and they send it to each other. Clara is listening to every message so she also receives the color. Let’s say the color is yellow. This is the “public” color.

Next, the two parties, Joe and Jack, independent of each other, come up with a private color. Let’s say Joe’s color is red, and Jack’s color is blue. They mix those two colors independent of each other and get a resultant color. Now Joe’s resultant color is orange and Jack’s resultant color is green. These new mixtures (orange and green) are now communicated with each other.

At this point, Clara, the hacker, has yellow, orange, and green. Clara thinks she is very smart, but what comes next may shock you.

Joe and Jack receive each other’s mixed colors, so Joe receives green from Jack and Jack receives orange from Joe. But next, both Joe and Jack, mix their private colors with the mixed color they received from the other party. Now the resultant mix is a reddish brown.

The key here now, is that the reddish brown was never sent over the wire. Clara has no way of recreating the reddish brown. But somehow, both Joe and Jack, have access to reddish brown. This reddish brown can now serve as the shared key.

I’ve explained this with colors, but in the real world, we use numbers. This is the foundation of RSA encryption, which forms the fundamental building block for many kinds of encryption algorithms in use today.

The technical word for what I just explained is “symmetric key exchange.”

## The Problem with the Solution: Asymmetric Key Exchange

Symmetric key exchange is impressive. It works, it’s secure, but it has a huge shortcoming. It requires the management of too many keys. Symmetric key exchange works fine if you have two parties communicating with each other. But what if you’re a website with many transactions and many people inputting data, such as a bank. And you have millions of browsers accessing you at the same time. Are you supposed to somehow securely manage the keys of all these millions of browsers? That’s millions of keys you’d have to manage.

Have you ever seen a padlock that you can lock by just pressing it down but to unlock the padlock you need a key?

This the foundation of our solution. In computer algorithms, you have an equivalent of this padlock. You have public private keypairs, typically a certificate. The idea is that anyone can close the lock with the public key, but to unlock the lock, i.e., decrypt the message, you need the private key.

So the bank now has a public-private keypair. The private key, the key to the padlock, never leaves the bank. But anyone who wishes to send a message securely to the bank can request the public key. Now the sender can encrypt a message with the public key, i.e., lock it, and send back the locked padlock, i.e., the encrypted message, back to the bank. The bank can now unlock the message with the private key that was never sent over the wire.

## Hybrid Solutions

It looks like asymmetric key exchange offers the best solution here, but not so fast. The first problem with asymmetric key exchange is that it’s computationally expensive. The second problem is that you have to keep that private key secure. And if such a key leaks, you’d have no idea it has leaked.

There are other issues as well, such as brute-force attacks, but most modern certificates use a key long enough to make such attacks impractical. There is, of course, the risk of man-in-the-middle attack, an intermediary who simply swaps out public keys and pretends to be the recipient (the bank in the example) and forwards messages while reading them. This attack is also mitigated via mechanisms that verify the authenticity of the sender and receiver. I’ll talk about this shortly when I talk about TLS.

Most issues are mitigated except keeping the private key secure and the computation expense problem.

The computation expense problem can be mitigated by exchanging a shared key for a symmetric key algorithm using asymmetric key exchange. This way, you pay higher up-front computation costs for only the initial handshake or whenever you renegotiate the key. Subsequent communication occurs using symmetric key exchange. As I’ll explain later, the computation expense problem can also be mitigated by using modern algorithms and specialized hardware.

As far as private keys getting compromised, well, that’s up to you to keep them secure. But if they do get compromised, there’s a mechanism via which a central certificate authority can revoke such an issued cert. This is called a CRL or certificate revocation list that the certificate-issuing authority must maintain. You can imagine that, over time, this CRL list gets huge and unwieldy, especially when it needs to be reliably communicated over the internet. This is why certificates have a validity: They automatically expire after a certain duration. CRLs greater than a certain age don’t need to be maintained.

There are also certificate trust chains, using which you can revoke a group of certificates easily, but I’m getting ahead of myself.

This does make me scratch my head a bit. Imagine a soldier in Afghanistan who needs to swipe a smartcard to gain access to a secure resource, perhaps something that launches a deadly weapon. In the middle of a war zone, with very poor connectivity over satellite, how can you be sure that the CRL at the edge location is updated and trustable? Imagine you’re under enemy fire and not being able to respond because IT can’t find their head in a bucket.

The real world is messy! In a perfect world, you can guarantee perfectly secure solutions. But in the imperfect world we live in, you always end up taking shortcuts for convenience and practicality. It’s these shortcuts that bite us in the long run.

It’s the shortcuts that bite us.

How do you check for a CRL in a submarine? How do you verify the authenticity of a message to fire nukes at an enemy in a submarine? Did you know that Lieutenant Colonel Stanislav Yevgrafovich Petrov was faced with a similar dilemma in 1983 when the nuclear early warning radar of the Soviet Union reported the launch of an ICBM with four missiles behind it coming from the United States. Deep inside a submarine, Lieutenant Colonel Stanislav Yevgrafovich Petrov had to decide whether this message was false and risk the annihilation of the USSR, or decide that this message was true, respond, and risk the destruction of the planet. Out of pure gut feelings, he decided this was a false alarm and effectively saved the world. Unfortunately, for this, he faced intense questioning by his superiors and was reprimanded.

It's really worrisome how many times since 1945 the world has come close to catastrophe, and how national leaders treat this with such nonchalance. It’s frankly a miracle that we’ve made it this far without an accident we weren’t able to manage.

In my daily mundane life, I really wonder sometimes, sitting in meetings where we make security decisions driven by budget, convenience, and adoption, if we are making the right decision.

Anyway, I digress. Let’s get back to hybrid solutions, a great example of which is TLS, something you use every day.

## TLS

TLS stands for transport layer security. It’s a protocol designed to provide communications security over a computer network. You probably use it every day when you check your email, when you open your browser, when you IM anyone. All of them use TLS. TLS guarantees the four tenets of cryptography, confidentiality, non-repudiation, integrity, and authenticity.

At the heart of TLS and many other things is PKI, or the public key infrastructure. PKI is a mechanism to create, manage, distribute, store, and revoke digital certificates across any network. For instance, on the internet, we have a PKI infrastructure. Your local company’s network may choose to set up its own PKI and CA (certificate authority).

Typically, when your browser visits [www.google.com](http://www.google.com), how can it trust that it’s the real Google? Google’s web server presents its certificate to the browser. And the browser must be able to trust that certificate. How can your browser trust Google’s certificate? The idea here is that Google’s certificate is issued by some authority that your computer already trusts. And that certificate authority is backed by its own certificate. It’s possible that the certifying authority isn’t trusted directly by your computer, but your computer trusts another authority that backs this certifying authority. This chain of trust can go a few levels deep, until eventually the site (Google’s) cert is backed by some authority you already trust.

Browsers typically show this information to you, as can be seen in **Figure 3**. This is a screenshot from Chrome on MacOS, but every OS and every modern browser has an equivalent of this.



Figure 3: Google’s certificate

If you explore the certificate a bit deeper, you’ll see that it’s backed by a certification path, as shown in **Figure 4**.



Figure 4: Google’s certification path and signature algorithm

Google has chosen to create its own internet authority, which may not be directly trusted by your computer. But it’s backed by GTS CA 1C3, which is backed by GRS Root R1, which is trusted by your computer. Therefore, you can now trust Google’s identity.

In the certificate, there are a lot of details. For instance, you can see that the certificate signature algorithm is PKCS #1 SHA-256 with RSA encryption. If you explore the fields further, you’ll find a lot of other interesting things, such as that this certificate can be used from a certain date and time until a certain date and time, that it has a thumbprint, that it can be used for proving identity and encryption, and many other interesting details.

### How Does This Really Work?

It all begins with the TLS handshake.

The client (browser), uses HTTPS to get to a location. The first message is a hello from the client, which contains details like the maximum TLS version the client can support, a random number, and a list of cipher suites they support. As of writing this article, TLS 1.2 is the most common version being used, TLS 1.1 or lower are no longer recommended, and TLS 1.3 is an improvement over TLS 1.3.

At this point, the server can look at the details and accept or refuse to communicate with the client. For instance, if the client insists on communicating with TLS 1.1 and the server deems it insecure, the server can refuse to communicate further. But let’s say that the server and client agree to use TLS 1.2 and a cipher suite they agree on.

Now the server responds with a message to the client informing the client about which TLS version the server will use, and pick one of the cipher suites and a random number.

Now the server sends a server key exchange message, which includes the public part of the cert. The digital signature is also included, which is again a hash of the previous messages, and signed using the private key of the server cert. Using the public key, the client can verify the digital signature. This is the proof that the server is who they say they are.

It’s worth adding that the server can also optionally request a certificate from the client at this point, which can prove the identity of the client to the server. This is common in certificate-based authentication.

Now the client replies with the client key exchange; remember this was the color example I gave earlier. This ends up resulting in a shared key secret.

Next, the client sends a change cipher message, which effectively means that I have the shared key, so let’s start encrypting. Using that shared key and the agreed upon cipher, the data is interchanged in an encrypted fashion.

## Encryption Algorithms

As is evident from this article, security continues to evolve. It’s worth getting familiar with some common encryption algorithms and their pros and cons.

As far as symmetric encryption algorithms go, there are three you need to know about: DES, 3DES, and AES.

The first is DES symmetric encryption algorithm. This is one of the most ancient encryption algorithms, introduced in 1976. It uses a 56-bit encryption key. DES is no longer considered secure and can be easily cracked using modern computers. A better replacement is either AES or Triple DES.

3DES or Triple DES is a better version of DES. 3DES applies the DES algorithm thrice to each data block. As a result, this process makes 3DES much harder to crack than its DES predecessor. Even so, 3DES, although better than DES, is also not considered secure. In fact, TLS 1.3 explicitly excludes the use of 3DES.

AES symmetric encryption algorithm is probably one of the most commonly used algorithms today. Unlike DES, AES is a family of block ciphers that consists of ciphers of different key lengths and block size. This makes it both safer and faster than DES. Today, you’ll see AES in use in many places, such as WiFi networks, VPN, SSL, and so many more.

As far as asymmetric encryption algorithms go, the two most popular algorithms in use today are RSA and ECC.

RSA was invented by three smart people: Ron Rivest, Adi Shamir, and Leonard Adleman. It uses prime factorization as its basis. This method involves two huge random prime numbers and these numbers are multiplied to create another giant number. The puzzle here is to determine the original prime numbers from this giant-sized multiplied number. This is a very hard problem to solve. To give you an idea, to crack an RSA-768 bit key, it would take 1500 years of computing time for all computers that exist on the planet today. The common standard is RSA-2048, which would take 300 trillion years. RSA-4096 would take longer than the entire age of the universe. The higher the bits, the more CPU you will consume, but the security gets exponentially better. Maybe some brilliant scientist one day will come up with an ingenious algorithm to crack RSA, but it's been around since 1977 and hasn’t been cracked yet. It’s postulated that quantum computers will end up cracking RSA in another 20-30 years. Maybe some governments already have secret super powerful computers that can crack RSA in extreme circumstances, but it’s safe to say that even with a quantum computer, which, practically speaking, doesn’t exist, it takes a long enough time to crack RSA. A lot of governments, therefore, force tech companies to give up their private keys as a pre-condition to operating in their country. The reality is, though, for all practical purposes, that most governments cannot crack open sufficiently secured communication. Whether that’s a good thing or a bad thing is a topic I’d rather not get into.

Another popular asymmetric encryption algorithm is the ECC or elliptic curve cryptography algorithm. ECC is a recent development, and more and more systems are using it. It’s particularly popular in the Apple world. In ECC, a number symbolizing a point on the curve is multiplied by another number and gives another point on the curve. To crack this puzzle, you must figure out the new point on the curve. The mathematics of ECC is built in such a way that it’s virtually impossible to find out the new point, even if you know the original point. The key advantage of ECC over RSA is that it can offer a similar level of protection as RSA but with a much shorter key. You can increase the key length to gain even better security than RSA, but ECC’s ability to work with shorter keys means that it can offer you the same protection as RSA with much less computational power. With the advent of mobile devices, you can imagine why ECC has been so popular. Also, this means that websites load quicker because the initial key exchange can be done quicker.

Isn’t the world we live in amazing? Just 10 years ago, we were peddling Internet Explorer 6 with DES, which was totally insecure and slow as a Lada full of elephants going uphill. Now we have dedicated chipsets taking full advantage of secure and faster algorithms on much higher-speed networks.

Then there’s the real world, where so many very large organizations continue to make mind-boggling decisions about security. And so many world leaders talk about nuclear weapons like they understand what they’re dealing with. Please send them a copy of this article if you can.

## Summary

When I was at the helm of picking careers for my life, I chose to be a computer engineer. My main reason was that this field will never be boring, there’s so much to learn, and people use fun words like bugs to describe problems. As the field and I have matured, this has been a constant: This field of work has continued to evolve at a furious pace.

Let me ask you this today: What kind of airplane will we be flying in the year 2092? You probably think of something super futuristic. What kind of car will we driving, what kind of houses will we live in, etc. But let’s go back 70 years into history. The Boeing 747 flew in 1952. And 70 years later, we’re squished into the middle seat, being treated like dirt, in basically the same technology. The slight improvements we have seen are all because of technological optimizations, but no quantum leaps have been made.

In fact, I’m hard pressed to think of any field of engineering where we’ve made a quantum leap besides IT. And all the improvements other fields have made are on the back of IT. Look at medicine. The MRI machine is pretty amazing, but at the end of the day, it’s a fancy computer or was designed by fancy computers. Look at rockets. They’re hyper-optimized using computers.

Now think of the change the tech sector has seen in the past 10 years. What will the next five or 10 years bring? As long as we treat the planet responsibly, this is an incredible time to be alive.

I’m a curious person with many interests. No technology exists in vacuum and cryptography has invented and reinvented itself as a result of circumstances that the human race has been through. I thought it was fun to talk a bit about history, and relate the stories to the real-world. This was an incredibly fun article to write and it’s a topic I think a lot about. What do you think? Did you enjoy reading all these stories and their effect on the field of cryptography? Let me know.

Until next time, kdssb frglqj! (This is a Caesar cipher with a shift of three. See if you can decode it).

Sponsored Sidebar

Are You Writing Secure C# Code?

Hopefully you are writing secure code, but hopefully isn’t good enough, is it? Learn to develop robust and secure C# applications in our online, three-day, hands-on, Secure Coding for C# Developers course. Register today! <https://www.codemag.com/training>