Public Key Encryption and One of its Weaknesses

”Nothing is covered up that will not be revealed, or hidden that will not be known.” — God

Student ID: 200006295

Word Count: 1995

Introduction

For as long as humans have existed, secrets have too. And as long as there are things that are hidden, there will be people trying to expose them. And in the internet era, there's one primary way we handle secrets — encryption. Public key encryption is an encryption standard that is used literally every day by anybody using the internet. It's used when you visit a webpage secured by SSL, it's used when you send an email, it'll even be used when I upload this essay. You could easily say that public key encryption is the pillar of safety that the entire internet and by extension the entire modern world stands on. But is it really that secure? In the course of this essay, I will define and explain public key encryption, talk about some of the different techniques and get into the weeds of one of the biggest flaws it had. The flaw in question is a possible backdoor into a specific algorithm called the Dual Elliptic Curve Deterministic Random Bit Generator or Dual_EC_DRBG for short. Talking about this is important due to just how pervasive public key encryption is.

The History

Our story is essentially one of trust, and it begins in 2006, when the National Institute of Standards and Technology (NIST) ratified a specific pseudo-random number generator (PRNG) called Dual_EC_DRBG[1]. However much we like to talk about the mathematics and theory of public key cryptography, the fact is that these concepts are ultimately implemented by companies before they come to us normal people using them. They're cemented in open standards built around trust and mathematics. Most individuals and most computer scientists as well won't be able to delve deep into the implementations and ratify the rigour they provide. That's why companies and people implementing these standards have to do their best to make sure they provide secure encryption for the millions of people using the standards. This was all flipped on its head when NIST, one of the biggest names in cryptographic compliance ratified this PRNG. But before we can come to why this specific elliptic curve is a big deal, we have to understand the role random numbers play in public key cryptography.

Public Key Cryptography

Public key cryptography works on the basis of 2 keys[3]. A private key which is used to encrypt your data, and a public key which is used to decrypt your data.[1] Your private key is something that you never share while your public key is information that's accessible to the rest of the world. To make a public key cryptography system work, you need an algorithm that's easy to compute in one direction, but near impossible to reverse. These such algorithms are known

[1]Note that your private key can also be used to encrypt and your public key can be used to decrypt. This is especially useful when verifying the identity of the person sending the message

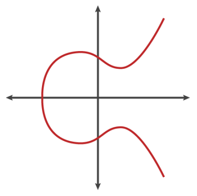
as “trap-door” or “one-way” functions. Finding a secure trap door function is essential to making a public key cryptographic system work. The first implementation of such a system came in 1977 with the RSA algorithm[3]. RSA essentially depended on the fact that multiplying two very large primes is much easier than factoring the product of two very large primes to find the two prime numbers that were originally multiplied. However this computation, while it was infeasible in 1977, soon proved to be not entirely secure. Algorithms like the quadratic sieve and general number field sieve were created and they were able to tackle the problem of prime factorisation with relative success. All of that meant that RSA wasn’t the elite cryptographic algorithm for the future. [2] So researchers came up with a better trap door function to suit their cryptographic needs.

Elliptic Curves

In the 1980s, researchers proposed a new trap door function based on a school of mathematics called elliptic curves. An elliptic curve is any set of points that satisfies the equation:

$$y^2 = x^3 + ax + b$$

It looks a little like this



and the nice symmetry you note isn’t just an aesthetic point in its favour: it also has some unique properties that make it good for public key cryptography[5]. The horizontal symmetry of the graph shows that any point of the curve can be reflected on the x-axis and remain the same curve. Another property to note is that any non-vertical line will intersect the curve in at most three places. If you take any two points on the curve and draw a line through them, the resulting line will intersect the curve in exactly one place. You can then reflect that point along the x-axis to get to the final point in the step. You can also use this operation (also called “dotting”) with one point over and over again. This is interesting because if you run this operation on any one point n times to arrive at another point, it’s very difficult to find out n when you only know the initial and final point. This is essentially the basis for an elliptic curve as a trap-door function. Trying to reverse this trap-door function is known as the elliptic curve discrete logarithm problem and for three decades mathematicians haven’t found an algorithm to reverse it other than a brute force approach.

[2]While RSA isn’t the best cryptographic algorithm, it is still very secure and is used in many places

Dual_EC _DRBG

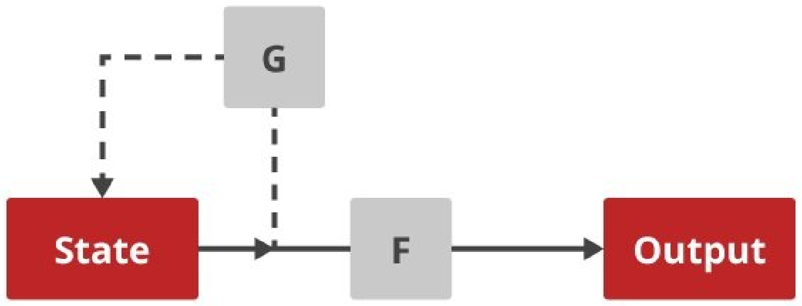
Coming back to the point of this essay: trust and this one specific PRNG. The Dual_EC_DRBG was introduced in Special Publication 800-90 by NIST and was later added to the international ISO standard. This same publication also proposed three new PRNGs, however, these differed from Dual_EC_DRBG as they mainly employed symmetric methods. Dual_EC_DRBG unlike the other three, turned out to be deeply flawed. Just like the generic example of elliptic curve cryptography above, Dual_EC_DRBG relies on the same principal of drawing lines and reflecting points. However, the proposers and promoters of the PRNG included some specific criteria that in hindsight look a bit suspicious.

The NSA and the origin of Dual_EC_DRBG

Although Dual_EC_DRBG was included by NIST in their specification in 2006, it was actually proposed years earlier by the NSA in another standardisation project at ANSI. The NSA has always been at the forefront of cryptographic innovation and cryptography standards in the US, and most of the implementations of cryptography derive from standards published by the US. After NIST included this PRNG in their specification, the NSA continued to champion it and promote it, something which seemed innocuous at first. Dual_EC_DRBG was never thought to be infallible however, problems with it were first described back in 2006 where it was proven that the random numbers it produces have a small bias. This problem on its own wasn't large enough to make the algorithm unusable, and the NIST standard even described a work-around for the issue. The bigger bombshell came in August of 2007, where Dan Shumow and Niels Ferguson of Microsoft showed a small 9 slide presentation at the CRYPTO 2007 conference[4]. Even though this presentation was small and informal, it caused waves of uneasiness in the crypto community, as it showed that one of the standards that was even being used by big companies such as RSA had a mathematical backdoor. Shumow and Ferguson however were quick to say that they weren't accusing NIST of intentionally inserting the backdoor, and the also chose not to mention the NSA at all in their presentation. What was incredibly suspicious however was the list of constants provided in the NIST specification used to define the algorithm's elliptic curve — no one could figure out where they came from. What's even more worrying is the fact that these constants acted as something as a public key for the algorithm; so whoever generated these constants could have easily generated a second set of number — a private key. Whoever generated these constants, could easily use their pre-existing knowledge to decrypt any information that used Dual_EC_DRBG. Essentially this provides whoever came up with the constants with a sort of skeleton key, some information that can be used to unlock anything encrypted by it. The final nail in the coffin came in 2013, when Edward Snowden released a smorgasbord of confidential documents from the NSA. These documents highlighted a variety of wrongdoings on the part of the NSA, but they also confirmed the NSA's role in the proposal of the Dual_EC_DRBG standard[6]. The internal NSA memos confirmed that that the NSA had worked during the standardisation process to eventually become to only editor of the Dual_EC_DRBG standard, and concluded that the NSA had indeed inserted a backdoor into the algorithm. The documents also confirmed that the NSA spend up to 250 million USD a year to insert backdoors into cryptographic standards[2]. This also solved the story of why so many companies had been using and promoting this algorithm — the NSA had paid them to.

The Flaw in Dual_EC_DRBG

Before I explain the mathematics behind the backdoor, I must first explain the basic construction of a pseudo-random number generator.



In this diagram, F and G are any independent trapdoor functions. In this construction, the internal state is kept secret, data is outputted through a one way function and the internal state is then updated by mixing the data back in with the state. If at any point the attacker can figure out the internal state, the can predict the output of the function. We can use an elliptic curve trapdoor function (like Dual_EC_DRBG) as F and G. Each function is hard to reverse, and if we choose their starting points (let's call them P1 and P2) randomly, they should be entirely independent of each other. So how was the backdoor added? Well the trick is to choose P1 and P2 so that they look random, but secretly have a special relationship with each other. We could choose P2 to be P1 dotted with itself s times over. Then P1 and P2 would be related. This is convenient because it looks random, and it's also very hard to prove since finding n requires solving the elliptic curve discrete logarithm problem. Given an initial state n, we can look at what the output is, and what s gets updated to.

The output of the x coordinate is:

$$Q = P1\, dot\, P1\, dot\, \ldots\, dot\, P1\, (n\, times)$$

Then the state gets updated to

$$State = P2\ dot\ P2\ dot\ \ldots\ dot\ P2\ (n\ times)$$

But since P2 is just P1 dotted with itself s times, the state is really

$$State = (P1\ dot\ P1\ dot\ \ldots\ dot\ P1\ (n\ times))\ dot$$

$$\ldots\ dot\ (P1\ dot\ P1\ dot\ \ldots\ dot\ P1\ (n\ times))\ (s\ times)$$

Since P1 dotted with itself n times is the output Q, we can write this as

$$Q\ dot\ Q\ dot\ \ldots\ dot\ Q\ (n\ times)$$

And since we know s and the output, we can calculate the next internal state of the algorithm. And since finding s otherwise requires solving the discrete logarithm problem, whoever chose the two initial starting points can be satisfied that they'll be the only person who can reverse the algorithm.

Conclusion

Given the evidence above, we can clearly see that Dual_EC_DRBG contains a backdoor, it's also evident that the insertion of said backdoor was a deliberate act, intended to compromise the integrity of the algorithm and provide a skeleton key to the NSA. This algorithm was also used as a standard by implementations provided by many companies such as RSA and was only recalled when the NSA document leak occurred. While public-key cryptography has other flaws: compute time, secure key changes and others — choosing a secure method for random number generation is easily the most important part of the whole process. Which brings me back to the point of this essay — trust. The biggest weakness that any cryptographic method has is trust. We have to trust the people who implement the standards we use across the world, and we have to trust the mathematics behind all of it. And this specific case, we can easily see that the trust was violated.

References

[1] Elaine Barker and John Kelsey. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators.* June 2015. URL: `https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final`.

[2] Glenn Greenwald. *Revealed: how US and UK spy agencies defeat internet privacy and security.* Sept. 2013. URL: `https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security`.

[3] Richard A Mollin. *RSA and public-key cryptography - researchgate.net*. 1947. URL: https://www.researchgate.net/publication/329543838_RSA_and_public-key_cryptography.

[4] Dan Shumow and Niels Ferguson. *On the Possibility of a Back Door in the NIST SP800-90 Dual Ec Prng*. URL: http://rump2007.cr.yp.to/15-shumow.pdf.

[5] Nick Sullivan. *A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography*. Feb. 2019. URL: https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/.

[6] Kim Zetter. *How a Crypto 'Backdoor' Pitted the Tech World Against the NSA*. URL: https://www.wired.com/2013/09/nsa-backdoor/.