A different kind of Crypto

Parker Schmitt

November 16, 2014

Contents

1 Introduction

I refer to this topic as payload cryptography, but perhaps that's the wrong term. Generally cryptography is used to hide messages so that they cannot be easily decoded. With payload crypto we only have to worry about automated detection (usually) and our goal is to hide the existence of any pattern. With cryptography the existence of the encrypted message is well known. It is perfectly legitimate to have an encrypted web portal, or have someone checking an imap server over SSL. However delivering malware—not so legitimate.

The main principle behind crypto algorithms is confusion and diffusion. In payload crypto confusion is the main goal. We don't care about cryptanalysis, usually with many payload crypters they use static keys so the existence of actual crypto does not necessarily exist. If crypto exists where it shouldn't AV can use the "nothing to hide nothing to fear" approach and give an alert. So why not have AV not even know it is crypto. In this case payload crypto is closer to steganography than cryptography.

Currently many payload crypters or obfuscators use crypto algorithms from our normal suite such as AES. It is a common tactic in android malware to throw the phone numbers over AES so that they can't be seen over the wire, however predictability can fall to tactics such as dynamic code analysis. When we are going up against automated analysis we do not necessarily need to defeat cryptanalysis we just need to be unpredictable; which is not a priority with standard cryptography as knowing data is AES encrypted is not all that helpful. However with dynamic code analysis knowing the existence of crypto where it doesn't belong is a red flag. The way I want to lay this out is to go through some of the principles of modern crypto, show its advantages and how it's implemented, then talk about classical crypto its disadvantages, and then talk about why classical methods are better for payload obfuscation and the advantages of modern crypto usually go away with payload obfuscation. Then we will talk about the best methods to hide payloads and how to be ahead of the curve with "signatureless" and signature based AV.

2 A brief discussion of modern crypto

Before explaining why the methodology for payload cryptography and cryptography for data protection/transport layer protection are different: we should go through how modern cryptography works to help understand the justifications behind modern crypto algorithms. In modern crypto the common tactic is to come up with an algebraic structure that achieves the goal of confusion and diffusion. The binary byte is usually moved into the new mathematical structure and then operations are done on that byte.

The goal behind this is so that it is hard to match each byte, or even bit in such a way that plaintext can be reverse engineered from ciphertext as usually an AES encrypted transmission has some known plaintext. When you are logging on to Amazon, everyone knows the Amazon banner is there, if knowing the

plaintext yields the key there is something majorly wrong.

2.1 How modern (non-payload) crypto works

With modern cryptography there are a two main elements, public key and shared key. Shared key crypto is more secure but there is no safe way to transmit the key. Public key is slower but can transmit data without leaking a key. Therefore the two encryption types are used in conjunction; public key to transmit the key and then use a shared key algorithm to transmit the data. The shared key algorithms are often used in conjunction with a method to convert a block cipher into a stream cipher otherwise almost all algorithms become vulnerable to cryptanalysis.

When trying to break a shared key algorithm usually the attacker guesses part of the plaintext and tries to make a mapping. Usually the plaintext is partially known to the attacker since there are known values and headers. Therefore there must be protection against this. With most private key crypto algorithms they are designed so determining a key knowing a plaintext and a ciphertext block simultaneously is not practical. Usually this involves permutation tables which is a process of mapping each bit. Here is the underlying problem with using standard block ciphers for payload obfuscation. The permutation tables are known. The permutations are necessary to map each byte to an element of a finite field.

With a modern shared-key cryptography algorithm there is enough work needed to encrypt or decrypt the data so that brute forcing is not possible and with a plaintext and ciphertext block the key cannot be determined. However a direct map between plaintext blocks and ciphertext blocks is possible. This is a well known attack on most block ciphers and an advantage of stream ciphers. This is where block cipher modes of operation come in which makes using a block cipher not just encrypting block by block with the key. (known as an electronic codebook)

The next challenge is how does one pass the key so it is not seen. This is where public key algorithms or key exchanges come in. There is either the ability to encrypt without knowing the decryption key or the ability to establish a key without sending the key in cleartext or using a static key. However there is one problem left. How do you know if the host you're communicating with, who sends you a key, is who they say they are? That is where certificates and signatures come in to play. Now what does this all have in common with payload obfuscation? Almost nothing. Payload obfuscation has one goal, to evade any analytics. An obfuscated payload will remain "encrypted" until it executes on the victim's machine and the decryption process should be subtle; these are mostly problems outside the scope of standard modern cryptography and that is why it is an entirely different problem.

2.2 Known Plaintext Attack

Let's first go through the RSA Algorithm. I will not be proving it as it is a known proof but it stems from Euler's theorem. Also I will be explaining the math but will be glossing over some aspects in the interest of not writing too much about the mathematical basis. We are going to define $\phi(n)$ to be the how many numbers less than n are relatively prime to n (or that they share no divisors besides 1, such as 2 and 5 can be divided by 1 and 2 and 1 and 5, they only share 1 in common therefore they are relatively prime). The ϕ function is hard to calculate unless you know the factors of n.

Factoring numbers is VERY hard, but you can know a large value of n and its prime factors simply by multiplying 2 large prime numbers together. ϕ has the property of being multiplicative so if p and q are prime

$$\phi(pq) = \phi(p)\phi(q) = (p-1)(q-1). \tag{1}$$

However if $n = pq$ then knowing n would not be enough to get you the ϕ function quickly unless you know p and q. This is the basis of RSA.

The next concept needed is modular arithmetic. Modular arithmetic is finding the remainder when dividing by a modulus such as $5 \equiv 1 mod 2$ since $\frac{5}{2} = 2 + \frac{1}{5}$

The first step is to choose two prime numbers p and q; they should be large. Let

$$n = pq. \tag{2}$$

Since we know p and q we can calculate $\phi(n)$ easily since they are prime numbers but without knowing p and q it's hard to calculate $\phi(n)$ Calculate $\phi(n)$ using (1). Choose e, the public key, such that

$$\gcd(\phi(n), e) = 1. \tag{3}$$

We do this so that we can have both a public and private key. Since we know $\phi(n)$ it's easy to calculate the private key which is why that value is as sensitive as d.

$$d \equiv e^{-1} \mod \phi(n). \tag{4}$$

We can calculate the plaintext and the ciphertext as follows.

$$m^e \equiv c \mod n. \tag{5}$$

$$c^d \equiv m \mod n. \tag{6}$$

The above algorithm is known as textbook RSA as there is no padding. Let's say someone implemented this as a csv with all the various plaintexts going to ciphertexts. Such that $m_1, m_2, \ldots, m_n \rightarrow c_1, c_2, \ldots, c_n$. Let's say Alice is sending a message to Bob using this implementation of RSA and Eve is listening. Eve has the public key e and n, as Bob had to send Alice the public key, and Eve has the ciphertext. Eve can guess the plaintext and if

$$m_{\text{guess}_n}^e \equiv c_n \mod n \Leftrightarrow m_n = m_{\text{guess}_n}. \tag{7}$$

This way Eve can know where she's right and where she's wrong. Usually this attack becomes practical when the cipher is used as a block cipher and parts of each block are known or if the entire message is encrypted and part of the plaintext is known it allows for easy ways for an offline brute force. However with payloads this does not matter, there are millions of signatures, and usually any form of cryptography will be disguised so it is nearly impossible to automate such cryptanalysis in dynamic code analysis.

3 Back to Basics: Reviving classical cryptography

Before cryptography was widespread the idea of coding a message was not too well known. Therefore cryptanalysis was not that common. Codebreaking was a very new art—as is dynamic code analysis. Therefore we should start with the basic goal, hide a message from detection. Encrypted messages were almost steganographic in nature as no one knew what was there.

The simplest and one of the first known ciphers is the Cesar cipher. It is the simple shift of letters such as a to c, b to d, e to g... Today that would be broken instantaneously, especially since we've seen it before. Mfu't ljkk Dftbs can quickly be decoded as "Let's kill Cesar". No one would use a cipher this simple for encrypted transmission of data but when trying to obfuscate data over the wire and not show the existence of crypto. The equivalent to such a cipher would be a one time pad, xoring every key by another byte (xor ($\oplus$) such as $1 \oplus 0 = 0$, $1 \oplus 1 = 0$, $0 \oplus 1 = 1$. Following the Cesar cipher there were two other classical ciphers (that translate easily into a computer-friendly version), substitution, and vigenere. Substitution is the cipher of cryptograms in the newspaper: each letter has a substituted letter. The other is the vigenere, where a word was determined as the encryption key and the Cesar cipher style shift is done for each letter, such as if the word was "apple" the first letter would be shifted by 1, the second letter by 15, third by 15, fourth by 11, the fifth by 5, and the sixth by 1 since it repeats the pattern. This can easily be done as an xor cipher too by xoring bytes instead of shifting letters. The substitution cipher can also be done with bytes by creating a shifting table.

4 Designing payload crypto algorithms

It is a lot easier to design crypto algorithms when they don't have to be good. The goals are very different in payload obfuscation; when encrypting data the fact that it is encrypted and the algorithm used is no secret. However when loading malware technically speaking you have no business sending encrypted data. If it is known to be encrypted, automated tools will use the "nothing to hide nothing to fear" since executables are rarely decrypted in ram (except may be with DRM).. Payload crypto needs to fool dynamic code analysis. Once you have arbitrary code execution it's all over. Once the payload is decrypted

the ciphertext can be removed and once the payload is running then there is no longer any need for payload crypto.

Why is it so much harder for a signature based antivirus/ids to detect 0days? Because they have not seen the attack vector before. Same goes for payloads. So why not have our payload crypto algorithm be an "0-day" every time. When the algorithm does not need to prevent cryptanalysis the requirements become as follows:

1. The file must be transformed into something indistinguishable from the original file
2. It must be "decryptable" preferably in RAM
3. The method used for decryption must be hard to detect and almost unknown.

The third item is the most important in this case. With any downloaded executable code it is going to be scanned by any security tool that can get its hands on it. If it is a binary that decrypts some random file that should set of an alarm so it must look like it's just reading a file and processing the data. Therefore in this case libraries should not be used and yes, it is actually preferred to implement your own crypto.

4.1 Encrypting data when Exfiltrating

So normally when making an encrypted tunnel one uses TLS or another standard. However encrypted data being sent outbound can be suspicious. (Unless it is standard https traffic, though sometimes companies use ssl-inspection; which is not necessarily a good idea but eliminates that path). So let's say an attacker is in a situation where he wants to exfiltrate data, or send data out, but there are no *standard* encrypted ways out.

If he wants to send data out he's going to have to find a way that avoids detection. If he sends the data in cleartext it may be detected; or the exact accounts (or whatever data is stolen) will be known as stolen and when stealing data it's always best to keep it a mystery as to what was stolen. If a standard crypto algorithm is used there can be ways of detecting it (there are not as many today but there might be in the future). However, usually an exfiltration takes a short period of time. The Home Depot breach was 2 weeks of SMTP outbound. Therefore cryptanalysis is not as much of a concern, but the ability to encrypt and send the data must be secretive. In addition it may be better to make the data look like something else. Therefore a substitution cipher may be a good option however, a substitution such as $0x00 \rightarrow Puppy, 0x01 \rightarrow Lab \ldots$ would work much better than dumping raw encrypted data. In this case the crypto algorithms are steganographic in nature.

Now let's look at a scenario with the requirement to stand up to cryptanalysis. In this case we will have to use a more standard crypto algorithm, I'd avoid AES and may be use a lesser known one, but we will have to use a more

mainstream algorithm. Though one thing that may be more easily detected is the key exchange. However, the key exchange can be done over a separate medium. Let's say we do a Diffie-Hellman over dtmf (dialtones) and then upload the encrypted data to the web. You can even do the configuration over another medium (such as dtmf) to obfuscate how the malware works. Obfuscation is your friend. In addition, I would use a classical cipher technique to obfuscate the data. Now the major weakness is the encrypting of the data since it's a known method, so I would not use any known libraries.

4.2 How to hide arithmetic

Remember when you first learned multiplication? How was it defined? Repeated edition so $4*3 = 4+4+4 = 12$ You can hide multiplication that way, what about exponentiation. That's repeated multiplication. Now to add to that, there are many ways to add, multiply, and exponentiate and many ways a computer can do it too. When efficiency is not the goal but obfuscation is; deoptimization is a viable technique. For example how many ways can one multiply by 8? I will list a few:

1. $n * (2 << 3)$ (where $<<$ is a bit shift)
2. $n + n + n + n + n + n + n + n$
3. $n << 1 * 4$
4. $n << 1 * 2 << 1$
5. $(2 * n)^3/n^2$
6. $n \oplus (n << 3) \oplus ((n << 3) >> 3)$

With a $\oplus$ operation it can be disguised by xoring by multiple values. $\oplus$ is communitive. This way xor ciphers can look like totally random xors on various bytes. However, be sure not to allow compilers to optimize your code to keep the obfuscation from being optimized away.

4.3 Generating "0-day" crypto automatically

Since the algorithm has far fewer requirements we can take a few principles and randomly generate our own algorithms, as opposed to just keys. Let's use as an example Textbook-RSA and xor padding. We will create an example of this:

1. $\oplus$ is communitive and associative, or in simpler terms order does not matter $a \oplus b = b \oplus a$ and $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
2. Using RSA we can exponentiate any block and it is always reversible $c = m^e$ and $m = c^d$.
3. Iteration can be done in any order when iterating over blocks

4. Blocks can be any size

The $\oplus$ operation is more efficient, the RSA is mainly there just to mix things up. Here are some steps to creating a new crypto algorithm:

1. Take all the bytes of the plaintext and make random substrings from various indices
2. Create a random set of bytes and xor them over each substring the bytes need to be the same size of the substrings
3. Take certain substrings of the byte array and raise them to a power mod n while knowing the other key
4. repeat these steps in no particular order
5. Randomize the arithmetic as shown in the previous section

The substrings of the bytearray randomizes the algorithm since different bytes are being xored, the RSA can be done at another time. Using these steps you get a different "algorithm" every time. You can randomize the way it's implemented each time too but this way it makes the decryption look like a random set of xoring and raising values to powers (and that arithmetic is disguised).

5 Using this in practice with malware execution

This is just one example of using this practice in a real *red-team* situation. The payload will consist of an executable and a file with random bytes that has been *encrypted*. The executable file just does xors and multiplications, so it passes the AV. It also grabs the file loads it into an array in memory (this is common so it will not set off any detections), decrypts the file, and then executes the decrypted file.

A more elaborate way to do this might be having the executable download files from, let us say, youtube comments. The Youtube comments will be *encrypted* and possibly use a substitution cipher to look like english words. Then the attacker will be able to control its bot via youtube comments. With the crypto algorithm changing every time, possibly through random number generation it would even be hard for a sandbox or dynamic code analysis anti-virus to detect such a payload.

6 The Takeaways

When determining a crypto method it is important to look at what the goals are. Using an algorithm just because there is a library for it might not be the best choice. Many people think AES will "hide" their payloads better, and may be it will to a theoretical degree, but the key will be on the victim's machine and so it defeats the purpose; and the antivirus doesn't do cryptanalysis, but

what it can do is detect AES. However if you use a simple one-time-pad, it may have a much harder time. While security by obscurity is never a solution hacking-with-obscurity can be a great strategy against automated tools.