Cryptography: The strongest link in the chain*

Levente Buttyán and Boldizsár Bencsáth
Laboratory of Cryptography and System Security
Budapest University of Technology and Economics
www.crysys.hu

IT security architectures that use cryptographic elements sometimes fail, but it is rarely cryptography to blame. The reason is more often the use of cryptography in an inappropriate way, or the use of algorithms that do not really qualify as cryptographic. High quality cryptography is in fact the strongest link in the chain, and there are good reasons for that.

Introduction

Cryptography is an area of great importance within the field of IT security. It provides algorithmic methods to protect information from unauthorized disclosure and manipulation during storage and communications. IT security would be cumbersome and much more expensive without cryptography, because if cryptographic mechanisms would not be available, then all storage and communication services needed to be protected by physical measures. In some cases, for instance, in case of wireless communication systems, protection would even be impossible without cryptography, as one cannot really control the access to radio channels by physical means.

While cryptography is important, it must be clear that it is not a magic wand that solves all the security problems in IT systems. Indeed, within the IT security community, there is a folkloric saying often attributed to Bruce Schneier, a well-known security expert: "If you think cryptography will solve your problem, then you don't understand cryptography… and you don't understand your problem." There are important areas, for example, operating systems security, where cryptography is not so helpful. Although, it is used here and there to solve particular issues, such as hashing passwords and encrypting files, it is not so well-suited to handle control flow security problems that are in the core of operating systems security.

Moreover, even when cryptography is the appropriate approach, a cryptographic algorithm alone is rarely sufficient to solve the entire problem at hand. In other words, cryptographic algorithms are not used in isolation, but instead, they are usually part of a more complex system such as a communication protocol, an authorization scheme, or an identity management infrastructure. Therefore, talking about cryptography without considering the environment in which it is used can be interesting from an academic point of view, but it is not sufficient to understand and solve practical IT security problems. We prefer, and in this article, follow a practice oriented approach: we discuss how cryptography is used in practice and why systems using cryptography sometimes fail.

* Published in the Hackin9 Extra magazine (ISSN 1733-7186), Vol. 8, No. 1, pp. 8-11, 2012.

Another folkloric proverb says that security is similar to a chain: it breaks at the weakest link. It turns out that cryptography is rarely the weakest link, and we believe that there are good reasons for this, which we discuss at the end of this article. Security systems involving cryptographic building blocks usually fail due to bad design and human errors. Even when the failure is attributed to the cryptographic building block, the real problem often stems from one or both of the following two mistakes:

(i) the "cryptographic" algorithm was designed by non-experts and/or it did not go through a thorough analysis, therefore, it only looked like a real cryptographic algorithm, but in reality, it is a crappy design, doomed to fail, and it should not be called a cryptographic algorithm in the first place;

(ii) the cryptographic algorithm is strong enough, but it is used in an inappropriate application environment or in an inappropriate way.

It is not really cryptography to blame in any of these cases.

Of course, mistakes can be made by cryptographers too, and there are examples for breaking cryptographic algorithms (e.g., collision attacks against the MD5 hash function). The point is that this happens far less frequently than the other two types of failures. Therefore, in the sequel, we focus on (i) and (ii), discussing some examples for those kinds of failures in real systems. At the end of the article, we explain why we believe that good quality cryptography is in fact the strongest link in the chain.

Examples for bad cryptography

There are plenty of examples in real life for the security failure of a system due to the use of low quality "cryptographic" algorithms. A prominent recent example is the failure of Mifare Classic chip cards, used extensively in the field of automated fare collection, due to the weaknesses in the Crypto-1 stream cipher [1].

In many cases, including that of Crypto-1, the "cryptographic" algorithm is a homebrewed design and it is kept in secret, such that it cannot be thoroughly analyzed by the cryptographic community. This "security by obscurity" approach, however, usually leads to failure. First of all, in most of the cases, the algorithm eventually becomes disclosed either by means of reverse engineering or by some unintended release of design documents. Moreover, once they have been disclosed, homebrewed "cryptographic" algorithms are often broken, mostly due to the inappropriate trade-off between cost, performance, and security made in the design. While design flaws could be discovered by independent analysis, due to the requirement of keeping the algorithm in secret, homebrewed "cryptographic" algorithms are not analyzed by independent experts. The designers, on the other hand, are biased towards low cost and high performance at the expense of security, due to market pressure and strong competition.

In some other cases, the "cryptographic" algorithm is simply designed by someone with little knowledge of the field of cryptography; it is made available for free, and then used by others, again with little knowledge of cryptography. As an example for this case, we present below a "cryptographic" algorithm designed for the encryption of the values of the parameters passed in URLs of links; this is intended for the prevention of disclosing

information about the internal structure of a protected web site. This "cryptographic" algorithm is part of the secureURL.php package [2]. We analyzed this package in the context of a penetration testing work that we conducted for request by a client. The client's web site used the secureURL.php package for hiding URL parameters, and what is more, the entire design of the site's defense architecture heavily depended on the assumption that URL parameters were properly hidden. Unfortunately, in this case, the use of bad cryptography created a false impression of security for our client: We broke the "cryptographic" algorithm of secureURL.php, and this also allowed us to successfully break into the client's system (with some additional work, of course). We reported the flaw in the secureURL.php package and provided a detailed description of the analysis in [3].

Decryption attack by known cleartext-ciphertext pairs

When the secureURL.php package is used, URL parameters are encrypted and optionally protected by a checksum, such that an attacker cannot observe the URL parameters and fabricate modified parameters. The process of parameter encryption in the secureURL.php package is illustrated in Figures 1 and 2**.**

The encryption mechanism used for encoding the URL parameters is based on XOR-ing the plaintext parameter string with the MD5 digest of a user defined secret key repeated as many times as needed to mask the entire plaintext parameter string. The corresponding lines from function crypt($text,$key) of secureURL.php are the following:

```
$key = md5($key);
...
($crypt .= chr(ord($text[$i]) ^ ord($key[$j]));
```

The problem with this "encryption" algorithm is that if an attacker can guess a plaintext parameter string, and observe its encrypted version, then he can compute the MD5 digest of the key which is used by the encryption operation. The calculation is done as follows:

```
$key[$i]= chr(ord($crypt[$i])) ^ chr(ord($text[$i]);
```

As the MD5 digest $key is used to "encrypt" the parameters in every URL of the site, once it is obtained, all other "encrypted" parameter strings can be easily decrypted. Note that the user defined secret key is not revealed, but it is not needed: only its MD5 digest $key is used to mask the parameter strings.

For a successful attack, the attacker needs to know at least one plaintext parameter string and its "encrypted" version. Encrypted parameter strings can be easily obtained as they can be directly observed in URLs. The attacker can obtain plaintext parameter strings by multiple ways. First of all, a page might contain some parameter names and values in plaintext accidentally, e.g., debug messages, programmer's notes, log files, etc. It is also possible that the attacker can simply guess some parameters, e.g., this should not be a hard task for a search function where a large part of the parameter string would be based

on user input. It is also possible that the web site contains known modules, such as open source web components. In this case, the attacker can examine the original program to reveal information about parameter strings. In any case, the attacker can easily check if the information found is correct.

Attacking the integrity protection algorithm

Integrity protection of URL parameter strings is based on a keyed CRC-32 function. A checksum value is computed from the "encrypted" parameter string and the same MD5 digest of the secret key which is used for "encrypting" the parameter string, and it is appended to the end of the "encrypted" parameter string before base64 encoding. The checksum is verified by the web server before decrypting the "encrypted" parameter string, in order to detect any manipulations by an attacker. The corresponding PHP code for the keyed CRC-32 calculation in function hash($text) is the following:

```
return dechex(crc32(md5($text) . md5($this->key)));
```

It follows, that if the MD5 digest of the secret key is known to the attacker, then he can compute a proper checksum for any fabricated parameter string. And as we have shown above, an attacker can obtain the MD5 digest of the secret key, therefore the attacker can easily fabricate "encrypted" parameter strings with a proper integrity checksum.

Extensions

As we have seen, the "encryption" algorithm uses only the MD5 digest of the secret key, which is 32 characters long. Therefore, for recovering the whole MD5 digest, the attacker needs a plaintext parameter string whose length is at least 32 characters, and its "encrypted" version. If the attacker can obtain only a shorter plaintext parameter string, then he can still try to figure out the missing characters of the MD5 digest of the secret key; here are some possible methods:

- The attacker randomly guesses the missing bits of the digest, calculates the integrity protection checksum, and compares it to the observed checksum value. If they do not match, then the guess for the missing bits was wrong, and the attacker repeats the procedure with a new guess.
- In another method, the attacker guesses the missing bits of the digest, creates an encrypted and integrity protected parameter string, and passes it to the server. If the server denies responding, then the guess was probably wrong.
- If there is no integrity protection, then the attacker guesses the missing bits of the digest, generates a falsified parameter string, sends it to the server, and checks the response. If the result of the query is not what is expected, then the guess of the bits was wrong.

Note, that the individual characters of the MD5 digest contain only 4 bit of information per character. In other words, every single character represents only a nibble (4 bits) of

the hash. This makes brute force attacks easier and it makes other types of attacks possible as well.

How to fix the problems of secureURL.php?

As we have seen, there are multiple problems with the encryption and integrity protection algorithms of secureURL.php, and we have not even mentioned some additional problems, such as the linearity of the CRC function with respect to the XOR encryption, which makes malicious modifications of an existing encrypted parameter string possible. Fixing the scheme is not straightforward, but some hints can be easily given: For the encryption algorithm, the usage of a state-of-the-art block cipher (e.g. AES-128) would help against the attack presented here, however, it would create longer encrypted URLs due to the fixed block size. For calculating the integrity protection checksum, CRC-32 should be avoided as it is not cryptographically strong. Instead, HMAC (with a strong hash function such as SHA-256) or some similar message authentication function should be used. Careful design is needed also for key handling, e.g., different keys should be derived and used for encryption and integrity protection.

Examples for good cryptography used in bad ways

Using a strong cryptographic algorithm as part of a more complex system does not by itself guarantee that the system will be secure: good cryptography can be used in bad ways. An entire family of examples for this is provided by the field of key establishment protocols. The objective of key establishment protocols is to setup a shared secret key between two (or more) parties in such a way that only the intended parties (and perhaps some other trusted party) learn the value of the key and they are convinced that the key is fresh. Such protocols typically use cryptographic building blocks to ensure the secrecy and the freshness of the established key. Many protocols were proposed in the 80's and 90's in the academic literature (see for example [4] for a comprehensive discussion), however, most of them turned out to be flawed some time after their publication. And the flaws were typically protocol flaws, not any weakness in the underlying cryptographic algorithm used in the protocol. Indeed, when analyzing key establishment protocols, researchers typically make the assumption that the underlying cryptographic building blocks are perfect, and they view them as black boxes.

While key establishment protocols provide a reach set of examples for how good cryptography can be used in bad ways, this set is biased towards academic examples, with few instances used in real systems in practice. This does not mean that practical systems lack good examples of this kind. Perhaps the most well-known case where an entire real system failed due to using good cryptography in bad ways is WEP, the first security architecture for WiFi networks. In fact, we use WEP as *the* example of how insecure systems can be built from relatively strong elements in our university lectures on network security at the Budapest University of Technology and Economics (lecture slides are available at www.crysys.hu). While the RC4 stream cipher used in WEP has known weaknesses, it can be used in a proper way, but the designers of WEP did not do so. The short IV length, and even more, the use of a stream cipher within a challenge-response

authentication protocol and for encrypting a CRC value for providing message integrity protection ruins the security of the entire system; and again, it is not really RC4 to blame, the real problem is in the way it is used in the WEP protocols (see [5] for a more detailed description of these problems).

Yet another example is the padding oracle attack on block encryption in CBC mode discovered by Serge Vaudenay and his colleagues [6]. One can view CBC mode as a low level protocol for encrypting large messages with a block cipher (which has a short and fixed input size of typically 16 bytes). Before encrypting a message in CBC mode, it has to be padded such that its length becomes a multiple of the block cipher's input size. When decrypting, this padding is removed at the receiver side, as it is not really part of the message; it is just added for technical reasons in order for CBC encryption to work. In many practical systems that use CBC mode, after decrypting an encrypted message, the receiver checks the format of the padding: if it is correct, then it is removed and the processing of the message is continued; otherwise some error is signaled. It turns out that the leakage of this one bit of information (i.e., whether the padding is correct or not) can be exploited to decrypt an entire encrypted message by repeatedly feeding the receiver with carefully constructed ciphertexts and observing the receiver's reaction (see [6] for details). Practical systems that use CBC mode and signal the result of padding verification include the SSL and the IPsec protocols, which are widely used in real installations. As with the other examples in this section, the flaw does not stem from the underlying block cipher, but rather from the way it is used in practice in CBC mode.

Why good cryptography is the strongest link?

In the previous sections, we argued that security failures mainly happen due to either the use of badly designed "cryptographic" algorithms, or the inappropriate use of strong cryptographic algorithms. Breaking a strong cryptographic algorithm happens only very rarely. In our view, the main reason for this is that, by today, cryptography has matured as a scientific field with very strong mathematical foundations. The cryptographic community has worked on different mathematical models in which various security properties can be precisely defined, and cryptographic algorithms can be analyzed in a rigorous manner. New cryptographic algorithms cannot be published today at prominent conferences of the cryptography community without a proof of security in some accepted model.

The design of more complex systems and security architectures lacks this sort of strong foundations. Although, some work has been done to construct formal models for encryption modes [7] and key establishment protocols [8][9], they are less developed and not routinely used. For larger systems, such as entire communication protocols or IT architectures, models and analysis techniques are even more in an under-developed state: existing models and tools are usually unable to handle the complexity of a practical system. We see this as a great challenge for the future in security research.

Another key point is that the cryptographic algorithms that are standardized and used on a wide scale undergo a very thorough selection process where competing proposals are

evaluated by independent experts with respect to different requirements. Such a thorough and open analysis before deployment is totally missing in case of larger systems and mostly in case of communication protocols too. An open source approach would mitigate the problem, but it contradicts the business policies of many companies.

To summarize, the strong foundations and the careful selection process make cryptography the strongest link in the chain. This should not come as a surprise, though: it is well-known for centuries that a long lasting building should not be built on sand but rather on a rock.

Perhaps one remaining question is how to help the proper usage of strong cryptographic algorithms in protocols and larger systems. Well, first of all, protocols and security architectures of IT systems should be engineered by security experts, as this is the case with long lasting buildings. Experts have background and experience, they know better the state of the art, and therefore, they can at least avoid known pitfalls with larger probability. Of course, anyone can become an expert in security engineering; however, similar to any other profession, that requires hard work and a humble attitude to approaching problems. A good introduction to security engineering is given by Ross Anderson in his book [10]; we recommend it as a starting point for those seriously interested in the subject.

References

1. N. T. Courtois, K. Nohl, S. O'Neil, Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards, EuroCrypt 2008, Rump Session.
2. Nguyen Quoc Bao, Secure URL 2.0, www.phpclasses.org/quocbao_secureurl
3. B. Bencsáth, SecureURL.php design flaws, CrySyS Lab Security Advisory, http://seclists.org/bugtraq/2011/Sep/139
4. C. Boyd, A. Mathuria, *Protocols for Authentication and Key Establishment*, Springer, 2003.
5. J. Edney, W. A. Arbaugh, *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*, Addison-Wesley, 2003.
6. S. Vaudenay, Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS..., Advances in Cryptology – EUROCRYPT 2002 Proceedings, Springer 2002.
7. M. Bellare, C. Namprempre, Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm, Advances in Cryptology – Asiacrypt 2000 Proceedings, Springer 2000.
8. M. Bellare, R. Canetti, H. Krawczyk, A modular approach to the design and analysis of authentication and key exchange protocols, 30th Annual ACM Symposium on the Theory of Computing, 1998.
9. P. Y. A. Ryan, S. A. Schneider, *Modelling and analysis of security protocols*. Addison-Wesley-Longman 2001.
10. R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd edition, Wiley 2008.

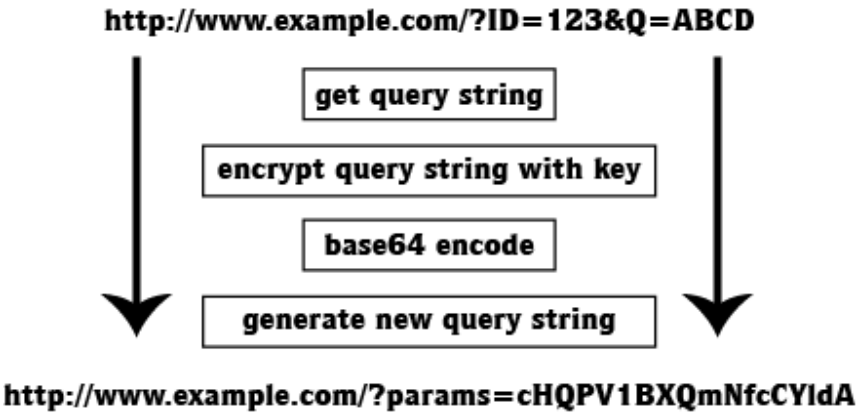

Figure 1: High level overview of the parameter string encryption in the secureURL.php package

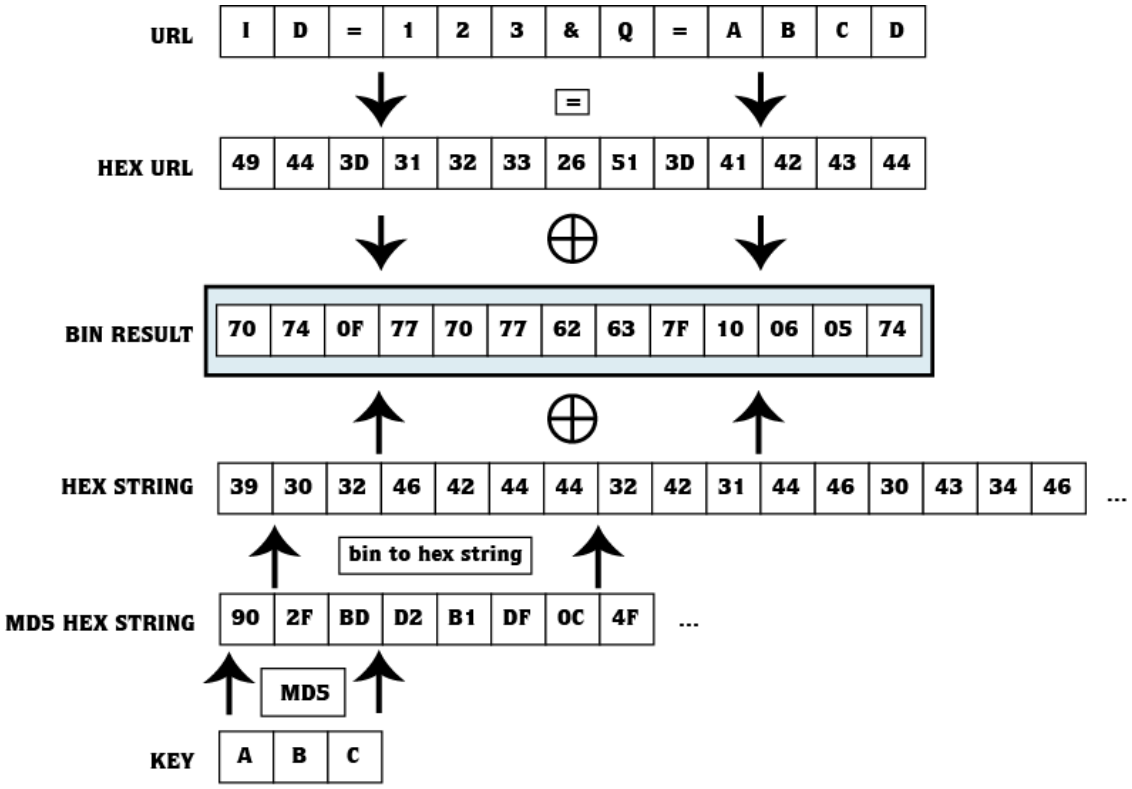

Figure 2: Details of the "encryption" algorithm of the secureURL.php package

Biographies

Levente Buttyán received the M.Sc. degree in Computer Science from the Budapest University of Technology and Economics (BME) in 1995, and earned the Ph.D. degree from the Swiss Federal Institute of Technology - Lausanne (EPFL) in 2002. In 2003, he joined the Department of Telecommunications at BME, where he currently holds a position as an Associate Professor and leads the Laboratory of Cryptography and Systems Security (CrySyS). His main research interests are in the design and analysis of secure protocols and privacy enhancing mechanisms for wired and wireless networks. Levente Buttyán has carried out research in various international research projects (e.g., UbiSecSens, SeVeCom, EU-MESH, WSAN4CIP), in which he had task leader and work package leader roles. He published several refereed journal articles and conference/workshop papers. He also co-authored a book on Security and Cooperation in Wireless Networks (secowinet.epfl.ch) published by the Cambridge University Press in 2008. Besides research, he has been teaching courses on network security and electronic commerce in the MSc program at BME, and gave invited lectures at various places. Levente Buttyán is an Associate Editor of the IEEE Transactions on Mobile Computing, an Area Editor of Elsevier Computer Communications, and a Steering Committee member of the ACM Conference on Wireless Network Security. He held visiting positions at EPFL and at the University of Washington, Seattle.

Boldizsár Bencsáth received the M.Sc. and Ph.D. degrees in Computer Science from the Budapest University of Technology and Economics (BME) in 2000 and 2009, respectively. He also earned the M.Sc. degree in economics from the Budapest University of Economics. From 1999, he is member of the Laboratory of Cryptography and Systems Security (CrySyS). His research interests are in network security, including DoS attacks, spam, malware, botnets, and virtualization. Boldizsár Bencsáth has led the team in the CrySyS lab that investigated the Duqu malware (this work received intensive press coverage). Besides research, he is currently responsible for the financial management of the Department of Telecommunications at BME, and he has also experience in the financial administration of EU and national projects.