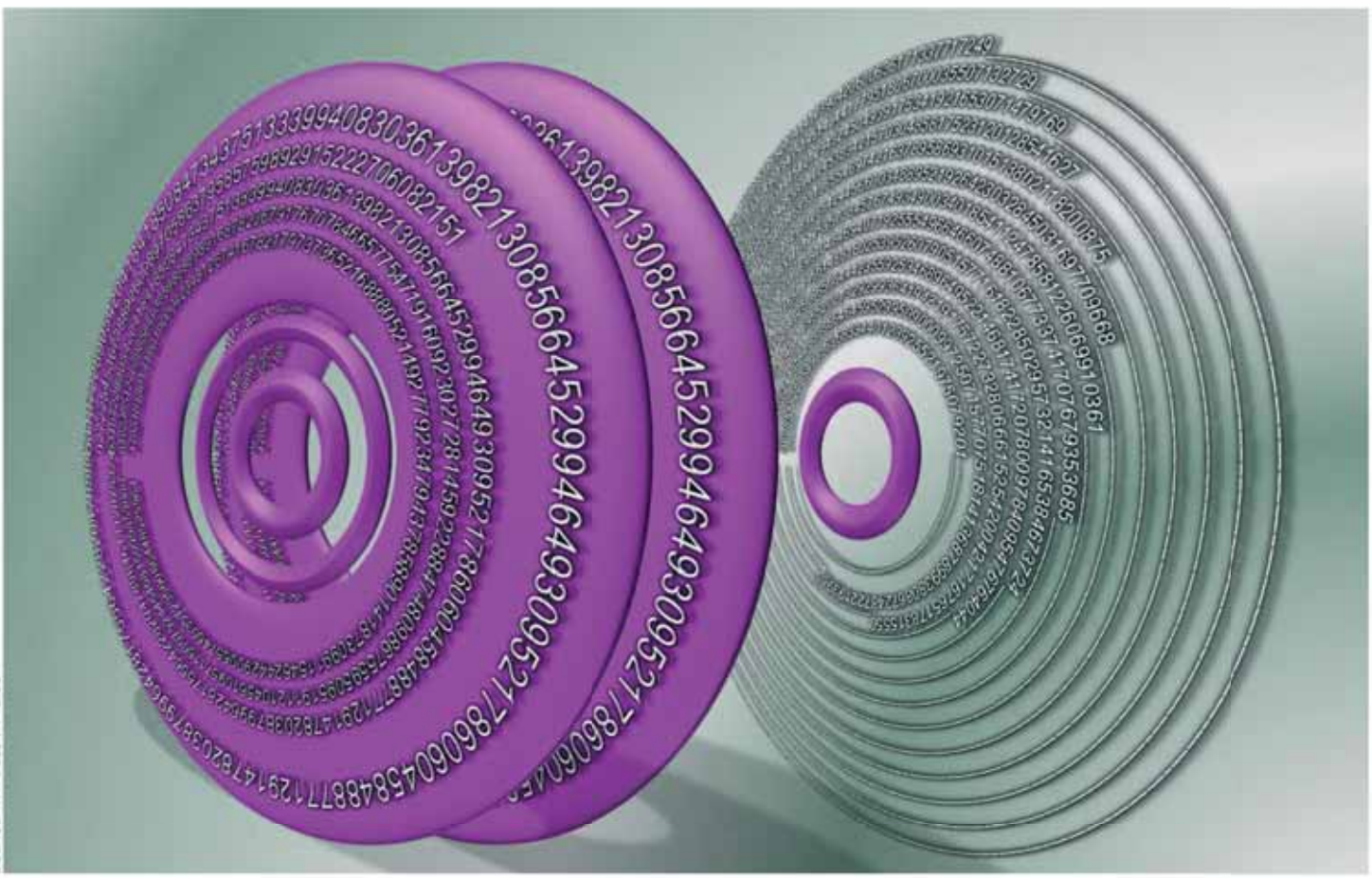
Hakin9 EXTRA

Issue 1/2012 (8) ISSN 1733-7186

Cryptography

The Strongest Link in the Security Chain



SOURCE: HTTP://EL.CRYPTO.COM

QKD FOR NEXT GENERATION NETWORKS

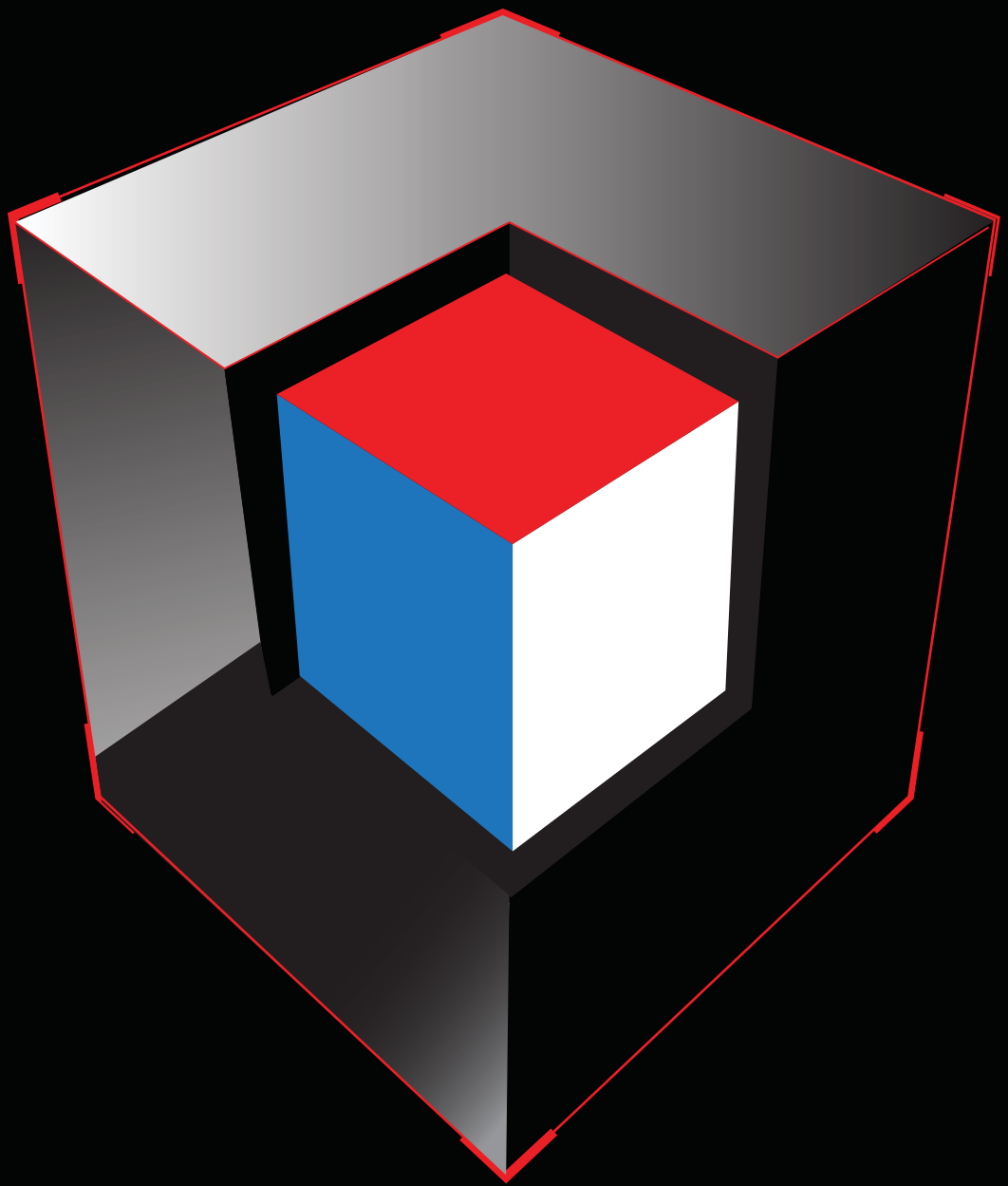
WEB APP CRYPTO

THE HASH FUNCTION CRISIS

SSL/TLS PKI ISSUES

PLUS

INTRUSION DETECTION AND RECOVERY FOR CYBER DEFENSE SYSTEMS



hitbseccconf2012

amsterdam

a m s t e r d a m

May 20th - 25th @ Okura Hotel Amsterdam

REGISTER ONLINE

http://conference.hitb.org/hitbsecconf2012ams/

Managing:
Michał Wiśniewski
m.wisniewski@software.com.pl

Senior Consultant/Publisher:
Paweł Marciniak

Editor in Chief:
Grzegorz Tabaka
grzegorz.tabaka@hakin9.org

Art Director:
Marcin Ziółkowski

DTP:
Marcin Ziółkowski
www.gdstudio.pl

Production Director:
Andrzej Kuca
andrzej.kuca@hakin9.org

Marketing Director:
Grzegorz Tabaka
grzegorz.tabaka@hakin9.org

Proofreadres:
Rebecca Wynn, Bob Folden,
Patrik Gange, Steve Hodge, Jonathan Edwards

Top Betatesters:
Thomas Stowe, Robert Wood,
Tim Thorniley, Scott Paddock,
Jeff Smith, Dan Dieterle, Michal Jachim,
Marek Janac, Jerome Athias,
Greg Tampa, Ayo Tayo Balogun

Publisher: Software Media Sp. z o.o. SK
02-682 Warszawa, ul. Bokserska 1
www.hakin9.org/en

Dear Hakin9 Extra followers

In the very beginnings of the year 2012 our team would like to wish you all the best in the New Year. We genuinely hope that you spent some wonderful moments with your nearest and dearest during Christmas and the New Year's Eve. We also hope that one of your New Year's resolutions is to become even more zealous Hakin9 Extra readers :-).

This issue is totally devoted to cryptography or, as I imagine it, the art of staying safe and invisible from any online threats and intruders' attacks. We do hope that thanks to our precious contributors you will get to know more about secure data transfer, encryption, decryption and authentication. This month's contributors will reveal how to remain safe and what are the best hints for not to get compromised.

In this issue, Levente Buttyán and Boldizsár Bencsáth will show you how important good Crypto really is. Travis H. is going to tell you why you should not mess with Crypto. Roberto Saia will present the fusion of mathematics, computer science and applied physics, a tripartite marriage of sorts. Speaking of marriages and home budgets – Zsolt Nemeth and Arun Sood will show you how to build cost-effective Cyber Defense Systems. As far as troubles are concerned, Bart Preneel will discuss Hash Function crisis and Martin Rublik will expose SSL/TLS PKI issues. Paul Baker will give you some tips on how to easily secure your data channels. Our futurists - Solange Ghernaouti-Hélie and Thomas Länger will discuss the prospect use of QKD (Quantum Key Distribution). Paweł Morawiecki will provide you with a tool for SAT-based cryptanalysis. Last but not least, an interview with Vadim Makarov by Nick Baronian is cherry on top of our Hakin9 Extra cake this month.

We hope that you enjoy reading Hakin9 Extra and come back to grab the next issue as soon as it appears online.

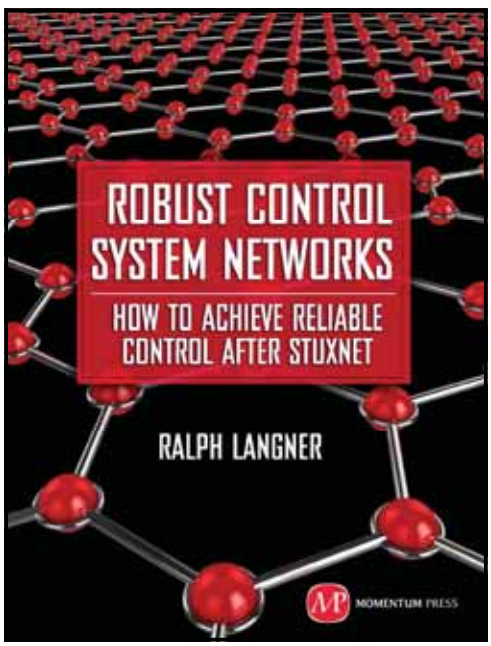
On behalf of Hakin9 Extra an Hakin9 team I would like to thank all the precious contributors, Beta Testers and proofreaders – without your effort and input Hakin9 Extra would simply not exist.

Stay Tuned!!!

Michał, Hakin9 Extra

P.S. There is a ciphertext hidden in this editorial note.

MOMENTUM PRESS



From the researcher who was one of the first to identify and analyze the infamous industrial control system malware "Stuxnet," comes a book that takes a new, radical approach to making Industrial control systems safe from such cyber attacks: design the controls systems themselves to be "robust."

***Ralph Langner** started a software and consulting company in the industrial IT sector. Over the last decade, this same company, Langner Communications, became a leading European consultancy for control system security in the private sector. The author received worldwide recognition as the first researcher to technically, tactically, and strategically analyze the Stuxnet malware.*

www.momentumpress.net
222 E. 46th Street, #203
New York, NY 10017

CRYPTOGRAPHY: THE STRONGEST LINK IN THE CHAIN

LEVENTE BUTTYÁN AND BOLDIZSÁR BENCSÁTH

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.

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.

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:

- 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;
- 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.

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. 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.

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 pa-

rameter 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 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 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. 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 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 and key establishment protocols , 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

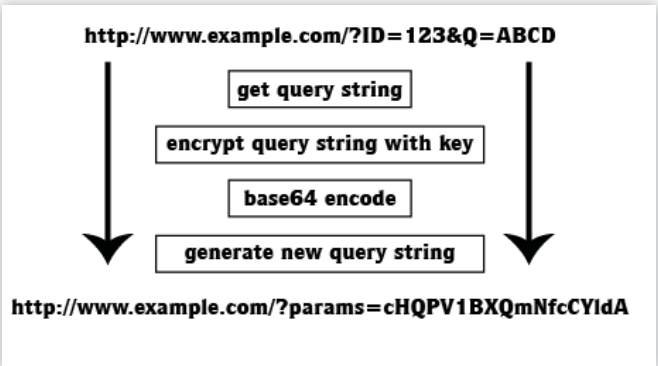


Figure 1: *High level overview of the parameter string encryption in the secureURL.php package*

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 ; 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.

LEVENTE BUTTYÁN AND BOLDIZSÁR BENCSÁTH

Laboratory of Cryptography and System Security
Budapest University of Technology and Economics
www.crysys.hu

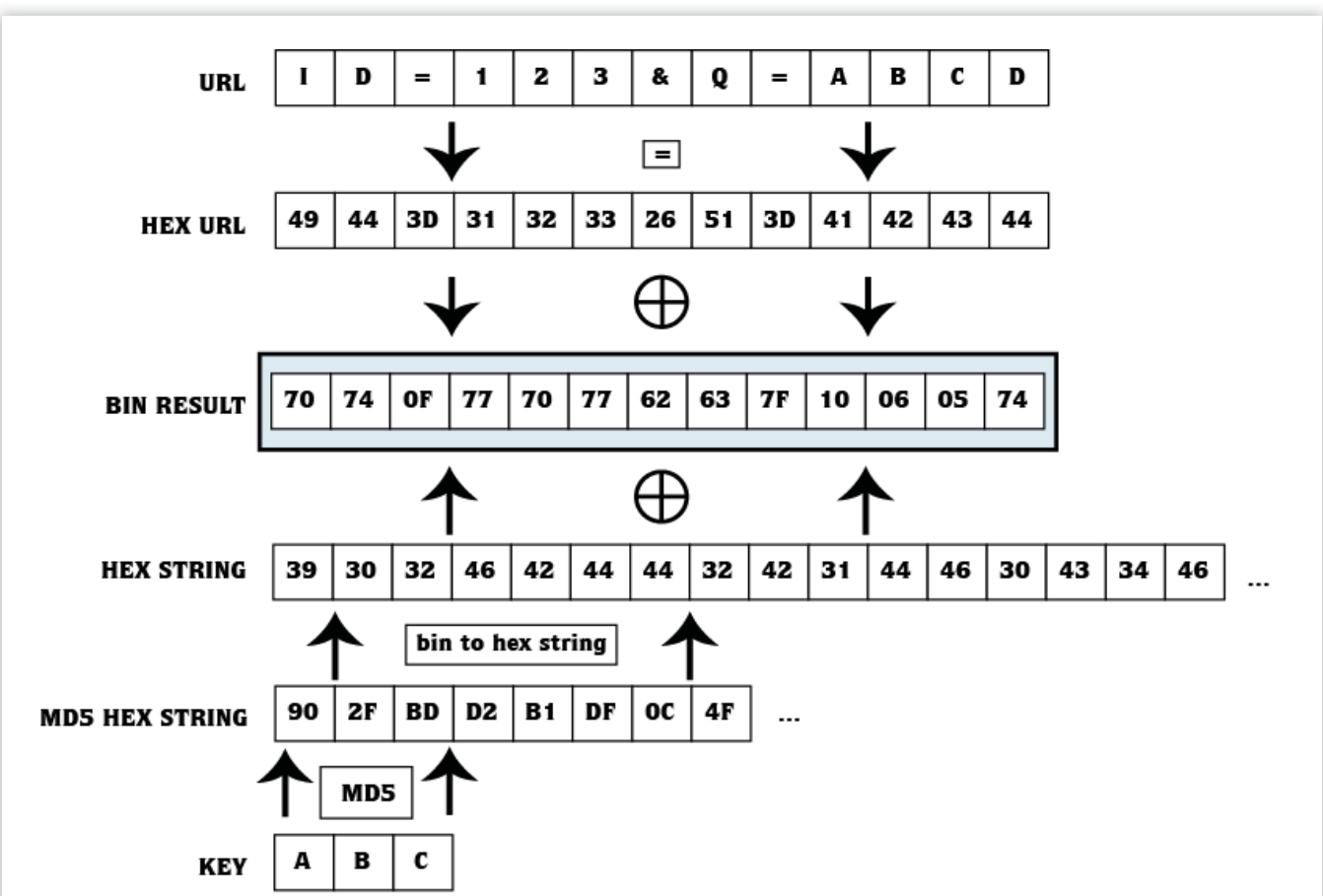


Figure 2: *Details of the "encryption" algorithm of the secureURL.php package*

COMBINING INTRUSION DETECTION

AND RECOVERY FOR BUILDING RESILIENT AND COST-EFFECTIVE CYBER DEFENSE SYSTEMS

ZSOLT NEMETH, ARUN SOOD

In this article we intend to show you a new approach in cyber intrusions. It has been behind the walls of ivory towers for years. Even in 2010 it was a nascent solution. Now it is roaming about with a few early adopters using it. What is this about and why is it worth it for them? By the time you finish this article, you'll find out.

We can easily agree that current cyber defenses are reactive and cannot protect against customized malware and other zero day attacks which we face today. Using Receiver Operating Characteristic curve analysis and damage cost models, we trade-off the true positive rate and false positive rate to compare alternative architectures. This analysis provides optimal value(s) of Probability of Detection by evaluating the potential damage from a missed intrusion and costs of processing false positives. In this article, we propose an approach which involves determining the influencing factors of each strategy and studying the impact of their variations within the context of an integrated intrusion defense strategy. Our goal is to manage the intrusion risks by proactively scheduling recovery using intrusion tolerance methods.

INTRODUCTION

The variety and complexity of cyber attacks are increasing, along with the number of successful intrusions to mission critical business systems. Recent breach reports like Wyndham Hotels [1] reported system compromise detection in February 2010, whereas the malware had resided in the system since October 2009. We could recite a lot more persistent intursions. So we infer that not only the Intrusion Detection System / Intrusion Prevention System (IDS/IPS) failed to prevent the adversary, but current systems were not able to detect the presence of the intruder long after the compromise.

Motivated by the above observations, more and more researchers are focusing on methods which consist of two important approaches to enhance cyber defense. First, recognizing that intrusion detection is a hard problem so that they can shift focus to minimizing losses resulting from intrusions. If this strategy is successful, they anticipate that the reduced demands on the IDS will in turn lead to fewer false positives. Second, their model uses real world data from recent breach reports and their average costs to evaluate the cost reductions that can be achieved by using a combination of intrusion detection and tolerance architectures.

Previously, the classical approach to assess architectures has been based on Single Loss Expectancy and Annual Loss Expectancy. More recently decision trees have been used [14]. In the former, many assumptions are required, and in the latter a lot of data has to be collected. These approaches are good for analyzing systems for which past data can be used. But is this a useful architectural for future decisions ?

We are proposing the use of ROC (Receiver Operating Characteristic) curve based analysis, which is a powerful tool system administrators can use with enterprise specific data to build economic models and to compare alternate architectures. The DARPA funded Lincoln Lab IDS evaluation [2] was a pioneering paper that evaluated many IDS by generating normal traffic similar to that seen on Air force bases. They used ROC curves to present their results. McHugh [3] published a critique of Lincoln Lab's work in 2000 which primarily considered issues associated with Lincoln's experimental dataset. McHugh pointed out the following problems in Lincoln's application of ROC analysis to IDS evaluation, which were a lack of "appropriate units of analysis, bias towards possibly unrealistic detection approaches and questionable presentation of false alarm data" [3]. In Section IV, we treat these issues.

In this article, we compare an IDS only solution with IDS and SCIT (Self Cleansing Intrusion Tolerance) combination, SCIT being our approach to intrusion tolerance which is classified in the recovery-based category [4]. From this assessment, optimal value(s) of Probability of Detection and other operational parameters can be selected to balance the potential damage from a missed intrusion and the cost of false positive process-

ing. In our approach, we stipulate that providing an upper bound on the time between the compromise and recovery has many advantages since it does not require the assumption that the system will be able to detect either the intrusion attempt or the compromise.

The rest of the article is organized as follows. In Section II, we develop the motivation for dependability recovery requirements. Section III briefly reviews the intrusion tolerance approach. Section IV, explains ROC Analysis usefulness to assess IDS architectures. . Section V, applies a cost model to evaluate how three different cases behave for a set of hypothetical ROC curves. Section VI is the conclusion.

MOTIVATION

As cyber defense efforts increase, passive efforts such as establishing anti-virus software, firewall protection, or improving password strength and encryption, the organization's workload is challenged by the need to apply patches immediately. Security researchers are uncovering close to 55,000 new malware samples a day, overwhelming malware analysis resources [5]. Increasingly, automated analysis technologies are used to keep up with the volume, but they still lack the precision to decipher compressed, encrypted, and obfuscated malware [6]. McAfee recent crash of tens of thousands of PCs globally illustrates the unpredictable system effects after compromise and their collateral damage, which creates even more uncertainty and less dependability for Enterprise Security [7].

The current reactive cyber defense approaches are expensive and inadequate. We expect that, automated recovery and Intrusion Tolerance Systems (ITS) will be useful in addressing the increasing malware and patch workload, but what are the cost impacts of malicious threats and false positives on dependability and security attributes?

INTRUSION TOLERANCE APPROACH

ITS architecture's objective is to tolerate unwanted intrusions and restore the system to its normal state. Various ITS approaches are reviewed by Nguyen and Sood [4]. In our paper, we use the recovery-based SCIT (Self-Cleansing Intrusion Tolerance) model [4], which is applicable to servers that are open to the Internet, such as Web, and DNS servers [8]. Using round-robin cleansing, at any point in time, a server in a SCIT cluster can have one of the three states: offline cleansing, offline spare and online transaction processing. The duration that a SCIT server is exposed to the Internet is called its Exposure Time. The architecture is simple, and does not rely on intrusion detection. Implementation of a SCIT scheme can be based on virtualization. The interfaces between controller and the group of servers to be protected are trusted.

Another benefit of a recovery-based ITS is it shrinks down breach duration, which has the effect of reducing losses and their costs. Indeed, this intrusion tolerance strategy would mitigate the effects of malicious attacks. Intrusion detection is known to be a hard problem, and current cyber defense systems reportedly detect less than half the malware. Still servers and apps account for 98% of the total record compromised. Verizon DBIR 2010 [9] underscores this problem by noting that only 11% of the compromises were detected within minutes or hours. Thus, current cyber defenses cannot protect systems against customized malware and other zero day attacks; once an attack is successful, it can persist for many weeks. This emphasizes the need for a recovery-based Intrusion Tolerance approach since a detection triggered ITS might again fall short of the needs.

RECEIVER OPERATING CHARACTERISTIC (ROC)

ROC analysis has long been used in signal detection theory to present the tradeoff between hit-rates and false-positive rates of classifiers. ROC analysis was initially used during World War II in the analysis of radar signals to differentiate signal from noise. It was soon introduced in Psychology to map the perceptual detection of signals [10]. ROC curves are useful for assessing the accuracy of predictions. A ROC curve plots the fraction of true positives (hits) versus the fraction of false positives, and hence has a direct relationship with diagnostic decision making. The ideal prediction method would yield a co-ordinate (0, 1) on the ROC curve. This represents 100 % true positives and zero percent false-positives, and is referred to as the perfect classification.

Using ROC to assess IDS quality.

The most attractive feature of ROC analysis is the fact that the trade-off between probability of detection and probability of false positive can be derived directly. This allows a system administrator to instantly determine how well a classifier performs and also to compare two classifiers. We care about false positives in addition to the probability of detection since there is a need to characterize the human workload involved in analyzing false positives generated by traffic. According to Lippman [2], false positive rates above 100 per day could make an IDS almost useless even with a high probability of malware detection since security analysts would spend hours each day investigating false positives.

DARPA funded Lincoln Lab IDS evaluation [2] appears to be the first to perform tests to evaluate many IDS by generating normal traffic similar to that on a government site. McHugh [3] reviewed and analyzed the validity and adequacy of artificial data used to estimate real world system performance. In this paper, we present a methodology to compare various IDS's, each of which is represented by a ROC curve. We utilize Verizon's 2010 results representing a cross section of multiple industries. Furthermore, these data validate firsthand real world evidence over a broad five year range from 2004-2009 with the addition of US Secret Service confirmed cases.

The Lincoln Lab experiment used ROC for presenting the results of the evaluation. McHugh [3] criticized Lincoln Lab's use of ROC curves primarily on the following grounds. We have attempted to address each of these concerns in our work:

- *Determining appropriate units of analysis*. Unit of analysis is the quantity of input on which a decision is made. Lincoln lab used sessions as the unit of analysis, the problems of which were outlined in [3]. McHugh also emphasized the need for using similar units of analysis across all IDS's to be evaluated. In our case, we consider a simple system and consistently use query / packet as our unit of analysis across all IDSs.
- *Errors per unit time*. In [2], a pseudo-ROC curve with x-axis as False Positives per day instead of Percentage False Positives was used. This led to two incomparable units being used on two axes, and the results in turn became strongly influenced by factors like the data rate that should typically be irrelevant. In this paper, we consistently use probability of detection and that of false positives for all ROC curves. In such a case, given that the distributions of signal and noise are realistic, McHugh [3] recognizes that

the ROC presentation should give a good account of detector performance in similar environments. Given enough characterizations of the signal and noise distributions, McHugh further acknowledges that it is even possible to investigate optimal detectors.

- McHugh [3] criticizes Lincoln Lab's methods of scoring and constructing ROC curves which lead to problems like bias towards unrealistic detection approaches, but not the use of ROC curves itself. In our case, the emphasis is not on constructing ROC curves but on comparing IDS's using our cost-model once we have their respective ROC curves. While there is a need for alternative taxonomies, the scoring method from the attacker's perspective is still utilized for real world incidents.

According to Lippmann, et. al. [2], there have been a number of similar efforts. In order to be able to compare multiple IDS systems, the ROC curves should be generated using similar or preferably same test data. According to Orfila et al. [11], if two ROC curves intersect at some point, there is no way of claiming that one is better than the other since some system administrators might want high probability of detection (top right corner of ROC curve) and some might want low probability of false positive (bottom left corner of ROC curve).

Stolfo et al. [12] presents an alternative method to perform evaluation based on cost metrics. Authors help formalize the costs involved in evaluating an IDS into three types: 1) Damage cost, 2) Challenge cost or Response cost and 3) Operational cost.

Drummond et al. [13] propose the use of cost curves for evaluating classifiers. Cost curves plot expected cost vs. Probability Cost Function (PCF). Here PCF is a function of probability of detection, probability of false positive and its corresponding costs. Although cost curves are good to compare classifiers, the representation does not provide for the system administrator to quickly see the cost trend of operating at different points (Pf, Pd) on the ROC curve. Also [13] does not suggest a way to determine the expected cost of operating at a point on ROC curve.

In [14], Gaffney et al. argued that both ROC analysis and cost analysis methods are incomplete. They used decision analysis techniques and provide an expected cost metric that reflects IDSs ROC curve based on a decision tree approach. This cost model requires a lot of data to be collected and does not reflect the magnitude of actual costs associated with breach events. For this, we propose a cost-model for the calculation of expected cost of operating at any point on the ROC curve.

V. COST MODEL

In this section, we look to overcome each of the shortcomings of earlier approaches by proposing a cost model that consists of two elements:

- A formula for the expected cost of operating at any point on the ROC curve
- Cost metrics derived from published breach investigation reports

A. Expected Cost calculation.

The cost of operating IDS at any point on the ROC curve (Pf, Pd) is a combination of the following:

- Operational Costs – Cost involved in operating the IDS and keeping it running.
- Damage Costs – the amount of damage caused by an intruder in case of a successful attack.
- Response Costs – the cost involved in responding to a potential intrusion on detection.

Out of the three costs mentioned above, operational costs and response costs greatly vary from organization to organization based on a number of factors like size of the organization, type of organization etc. Since these two costs are not entirely quantifiable, for the purposes of this paper, we employ the objective function proposed in [15].

Expected Cost of operating at any point on the ROC curve = Cost of Misses + Cost of False Positives.

Thus, for every point on the ROC curve (Pf, Pd), we have an expected cost:

- Expected Cost = (Cm*p*Pm) + (Cf*(1-p)*Pf), where
- Cm – Cost of a miss p – Prior probability of Intrusion
- Cf – Cost of a false positive Pd – Probability of detection
- Pm – Probability of a miss = (1-Pd)
- Pf – Probability of a false positive

Note that this expected cost is for one incoming query. If there are 'n' incoming queries, the above expected cost must be multiplied by 'n'. The value of metrics used in the cost model is summarized in Table 1.

In this paper, the probability of detection Pd and that of a false positive Pf will constitute the operational parameters.

We use the median number of records lost for assessing damage. In many cases, the outliers in breach data can skew the

Table 1. *Metrics Values Use in the Cost Model*

Metrics	Value	Explanation	Explanation
Median number of records lost per breach (M)	1,082	In cases of outliers this is a better representation of the "typical value"	In cases of outliers this is a better representation of the "typical value"
Average cost of a data breach per compromised record (D)	$ 204	Direct Cost: $ 60 Indirect Cost: $144	Direct Cost: $ 60 Indirect Cost: $144
Cost of a Miss (Cm)	$ 220,000	(Median number of records lost per breach) * (average cost of a data breach per compromised record) = 1082 * $ 204	(Median number of records lost per breach) * (average cost of a data breach per compromised record) = 1082 * $ 204
Cost of a False Alarm (Cf)	$ 400	Assumption: Labor Cost + Overhead Cost = $ 400	Assumption: Labor Cost + Overhead Cost = $ 400
Median Compromise (Duration per breach)	14 days	Median time spent from System compromise to Breach discovery + Median time spent from Breach Discovery to Breach Containment	Median time spent from System compromise to Breach discovery + Median time spent from Breach Discovery to Breach Containment

data, because most of the losses come from only a few breaches. Therefore, the Mean becomes highly skewed and is not a good estimate of the typical number of records lost per breach. Median is a better estimate of the typical value [16].

B. Evaluating classifiers using our Cost Model.
For the purposes of this paper, we do not address how the ROC curves are constructed. Proper construction and use of ROC curves in Intrusion / Anomaly detection have been addressed in [17]. We just show how the cost model can be implemented once they are constructed. Figure 1 gives a family of hypothetical ROC curves, each representing a classifier. We will implement our cost model on these ROC curves in three different cases to evaluate the classifiers' behaviors:

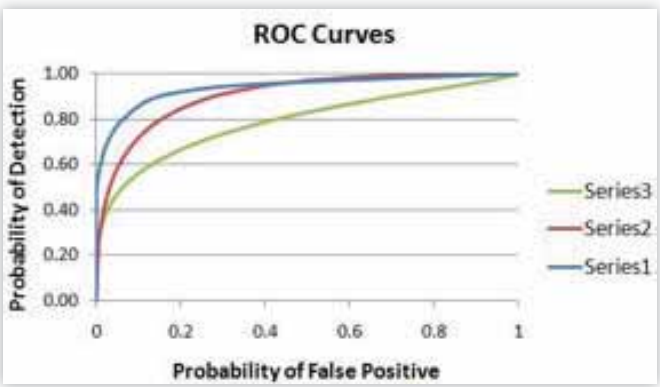


Figure 1. *Receiver Operating Curves*

Table 2 provides the values of the parameters used in the cost model in each of the three cases. Within each case, the value of 'p' remains the same for both IDS and SCIT+IDS. Therefore, the number of intrusions that occur in each of these architectures are the same since Number of intrusions = [Number of incoming queries * Prior probability of intrusion (p)]. The baseline IDS and SCIT+IDS scenarios are provided for Case 1. Case 2 and Case 3 help investigate the impact of 'Cm' and 'p' on system cost and security. Figures 2 through 7 illustrate this. It is noted that the y-axis scale is different in Figure 6.

CASE 1a. IDS: (Figure 2)
This is a stand-alone IDS system. The cost keeps decreasing as Probability of Detection (Pd) is increasing. As Pd increases, number of misses decrease along with the significant associated costs. However, after a threshold, if we keep increasing the value of Pd, the expected cost stops decreasing and starts increasing rapidly. At this point, the cost of False Positives exceeds the cost of misses and so the gains from containing misses start diminishing. This point is known as the "minimal cost point on the ROC curve (MCP)". For e.g., in Case 1a, the MCP for Series 1 is 70 and it occurs at (Pf, Pd) = (0.20, 0.85). MCP for each series of every case we evaluated is tabulated in Table 3.

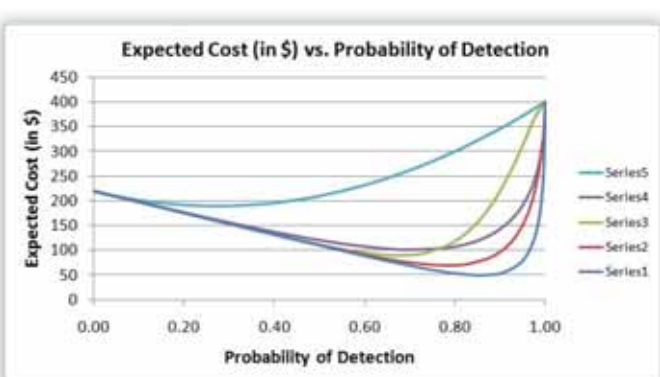


Figure 2. *IDS: Case 1a*

CASE 1b. SCIT + IDS: (Figure 3)
Now we add SCIT to existing IDS and evaluate the system using our Cost Model. We assume that the exposure time of SCIT is 4 hours1. This reduces the compromise duration of the system from 14 days to 4 hours. We assume that data is ex-filtrated uniformly over time. Since the cost of a miss was $220,000 earlier with compromise duration of 14 days, now it significantly reduces to $2,620 for compromise duration of 4 hours.

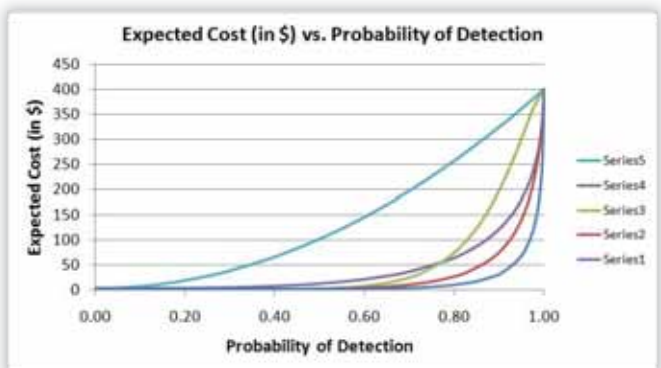


Figure 3. *SCIT + IDS: Case 1b*

CASE 2. (Figures 4 & 5)
Assumption: As compared to the baseline (Case 1), IDS cost of a miss is reduced from $220,000 to $60,000.

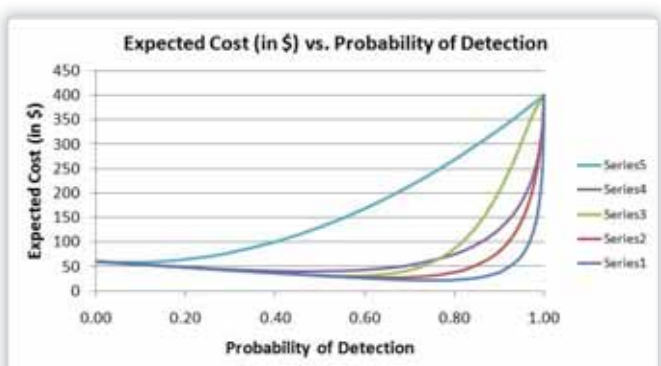


Figure 4. *IDS: Case 2a*

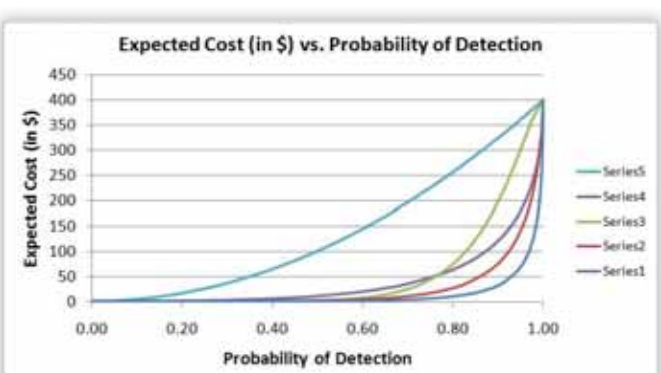


Figure 5. *SCIT + IDS: Case 2b*

CASE 3. (Figures 6 & 7)
Prior Probability of Intrusion is increased fivefold from p = 0.001 to p = 0.005.

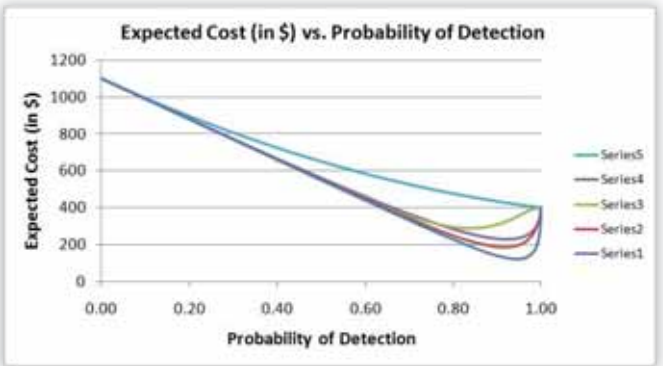


Figure 6. *IDS: Case 3a*

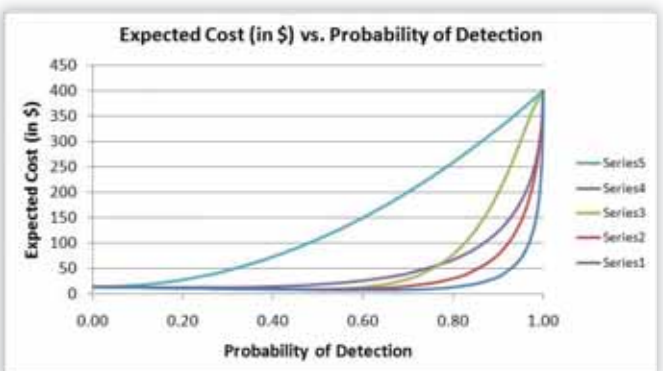


Figure 7. *SCIT + IDS: Case 3b*

C. Results: Comparison of IDS's.
Figure 8 compares the MCP's of 3 IDS' whose performances are indicated by the ROC curves in Figure 1.

- Series 1 IDS clearly outperforms all the other IDS in all three cases.
- It is most expensive to operate the IDS in case 3 since prior probability of intrusion is high which in turn leads to more misses.

D. Results: Comparison of SCIT + IDS's
Figure 8 also presents the minimal cost points for IDS + SCIT. We have used an exposure time of 4 hours. We note that as compared to the IDS only case, the costs are much lower. The minimal cost points are achieved using a much lower value of Probability of Detection which in turn leads to a lower Probability of False Positive. We conclude that this makes the IDS design much easier and the system easier to operate. The reliability of the IDS results also increase.

From the results, we can see that the benefits of adding SCIT are as follows:

- Cost of a miss is greatly reduced. As the compromise duration / exposure time of SCIT is reduced, cost of a miss further reduces.
- We can tolerate a larger number of misses now that the cost of a miss is reduced.

Roc curves

E. General Observations (IDS and SCIT + IDS)

- As the cost of miss decreases, we can tolerate more misses and so probability of detection for achieving minimal cost point can now take lower values.
- As Cm decreases, Cf has a greater influence on the expected cost and so there is an increased need to contain false positives. Note that the Probability of False Positives for achieving minimal cost point now decreases.

As prior probability of intrusion 'p' increases:

- The total number of misses increases and so does the expected cost.
- To combat this, probability of Detection for achieving minimal cost point increases thus reducing the number of misses. (Note: Number of misses = Number of incoming queries * p * Pm).

VI. CONCLUSION

Intrusion detection is a hard problem, making intrusions inevitable. Consequently, containing losses by an upper bound on the time between compromise and recovery shows many advantages. ROC analysis, supplemented with cost analysis using median of lost records and average cost of compromised records per breach, reveals tradeoff between high probability of detection, and low probability of false positive. Our approach reduces the cost of a miss;

Table 2. *Parameter values used in the cost model*

	P	C_m	C_f	Compromise duration
Case 1a: IDS	0,001	$ 220,000	$ 400	14 days
Case 1b: IDS + SCIT	0,001	$ 2,620	$ 400	4 hours
Case 2a: IDS	0,001	$ 60,000	$ 400	14 days
Case 2b: IDS + SCIT	0,001	$ 715	$ 400	4 hours
Case 3a: IDS	0,005	$ 220,000	$ 400	14 days
Case 4a: IDS + SCIT	0,005	$ 2,620	$ 400	4 hours

Table 3. *Minimal Cost Point Values*

CASE	Minimal Cost Point for Figure 1 ROC – Cost ($)					
	series 1		series 2		series 3	
	IDS only	IDS + SCIT	IDS only	IDS + SCIT	IDS only	IDS + SCIT
CASE1	70	2	102	3	135	3
CASE2	28	0,5	43	1	45	1
CASE3	170	7	218	12	386	12

and tolerating a larger number of misses' leads to lower false positive costs.

The SCIT architecture provides a robust security mechanism that guarantees certain security properties by limiting the exposure time. In addition, SCIT does not generate false positives and thus reduces the intrusion alerts management costs. Thus SCIT also provides administrative and economic benefits which make it a reasonable choice to be included in security architecture. In particular, this is expected to be of interest in environments where technical skills are limited. The analysis presented suggests that a combination of IDS with SCIT on host servers provides a robust architectural solution in the face of new attacks.

REFERENCES

[1] Hotchkiss, Kirsten. http://www.wyndhamworldwide.com/customer_care/data-claim.cfm. Jun. 2010.

[2] R. Lippmann, et al "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation" Proceedings of DISCEX 2000, Los Alamitos, CA. 2000.

[3] McHugh, John (2000) "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory" TISSEC, Vol 3, Issue 4

[4] Nguyen, Quyen and Sood, Arun. "Comparative Analysis of Intrusion- Tolerant System Architectures". IEEE Security and Privacy – Volume: PP , Issue: 99 , 2010.

[5] McAfee Labs. "McAfee Threats Report: Second Quarter 2010". http://www.mcafee.com/us/local_content/reports/q22010_threats_report_en.pdf. pg 11.

[6] Bejtlich, Richard. "The Tao of network security monitoring: beyond intrusion detection", Pearson Education, Inc. 2005.

[7] Kravets, David. "McAfee Probing Bungle That Sparked Global PC Crash".Threat Level. http://www.wired.com/threatlevel/2010/04/mcafeebungle/. 2010.

[8] Anantha K. Bangalore and Arun K Sood. "Securing Web Servers Using Self Cleansing Intrusion Tolerance (SCIT)", DEPEND 2009, Athens, Greece. 2009.

[9] Verizon Business Data Breach Investigations Report 2010.

[10] Swets. John A. "Signal detection theory and ROC analysis in psychology and diagnostics: Collected papers".

[11] Orfila, Augustin. Carbo, Javier. and Ribagardo, Arturo. "Advances in Data Mining, volume 4065, chapter Effectiveness Evaluation of Data Mining based IDS, pages 377-388. Springer Berlin Heidelberg. 2006.

[12] Stolfo, S. Fan,W. Lee, W. Prodromidis, A. and Chan, P. "Cost-based modeling for Fraud and Intrusion Detection: Results from the JAM Project" Proceedings of DISCEX 2000, Los Alamitos, CA. 2000.

[13] Drummond, Chris. Holte, Robert C. "What ROC Curves Can't do and Cost curves can". 2004.

[14] Gaffney, John E. Jr. Ulvila, Jacob W. (2001). "Evaluation of Intrusion Detectors: A Decision Theory Approach" Security and Privacy.

[15] J. Hancock and P. Wintz. Signal Detection Theory. McGraw-Hill. New York 1966

[16] Widup, Suzanne. (2010, Jul). "The Leaking Vault – Five years of data breaches" – Digital Forensics Association.

[17] R.A. Maxion and R.R. Roberts. "Proper use of ROC curves in Intrusion/ Anomaly Detection" Technical Report, University of Newcastle Nov 2004

[18] 2009 Annual Study: Cost of a Data Breach, Ponemon Institute LLC.

DR. ARUN SOOD

Dr. Arun Sood is Professor of Computer Science in the Department of Computer Science, and Co-Director of the International Cyber Center at George Mason University, Fairfax, VA. His research interests are in security architectures; image and multimedia computing; performance modeling and evaluation; simulation, modeling, and optimization.

He and his team of faculty and students have developed a new approach to server security, called Self Cleansing Intrusion Tolerance (SCIT). We convert static servers into dynamic servers and reduce the exposure of the servers, while maintaining uninterrupted service. This research has been supported by US Army, NIST through the Critical Infrastructure Program, SUN, Lockheed Martin, Commonwealth of Virgina CTRF (in partnership with Northrop Grumman). Recently SCIT technology was winner of the Global Security Challenge (GSC) sponsored Securities Technologies for Tomorrow Challenge. Dr Sood leads a university spin-off called SCIT Labs Inc, which is commercializing SCIT technology under license from GMU.

Since 2009 Dr. Sood has directed an annual workshop on Cyber Security and Global Affairs with Office of Naval Research support. The 2009 workshop was at Oxford, 2010 in Zurich and 2011 in Budapest. He was awarded grants by NATO to organize and direct advance study institutes in relational database machine architecture and active perception and robot vision. Dr. Sood has held academic positions at Wayne State University, Detroit, MI, Louisiana State University, Baton Rouge, and IIT, Delhi. His has been supported by the Office of Naval Research, NIMA (now NGA), National Science Foundation, U.S. Army Belvoir RD&E Center, U. S. Army TACOM, U.S. Department of Transportation, and private industry.

Dr. Sood received the B.Tech degree from the Indian Institute of Technology (IIT), Delhi, in 1966, and the M.S. and Ph.D. degrees in Electrical Engineering from Carnegie Mellon University, Pittsburgh, PA, in 1967 and 1971, respectively.

His research has resulted in more than 160 publications, 4 patents, 2 edited books.

ZSOLT NEMETH

Zsolt NEMETH is a serial entrepreneur who set up businesses in cyber security. His main interests are cryptography and network security. He founded MDS Ltd in the UK. He has done consulting and penetration testing for financial institutions and built up bespoke solutions for them. Meanwhile he was the leader of a team of cryptographers that worked on creating an elegant cipher that will potentially solve some of the significant issues of the Vernam Cipher (aka one-time-pad). After selling MDS Ltd he has founded a holding that has scouted, bought and licensed technologies out.

Now he runs Camphora Llc with offices in Hungary and Luxemburg. He is doing ethical hacking and intrusion analysis for SMEs and a few selected big companies.

Zsolt holds a Master of Science degree in Economics from Szechenyi Istvan University and a Master of Science degree in Applied Mathematics from Ecole National Superieure, Paris. He is fluent in Hungarian, French and English.

He is a frequent speaker at conferences on fast symmetric ciphers and SCADA systems security.

FROM THE THEORY OF PRIME NUMBERS TO QUANTUM CRYPTOGRAPHY

ROBERTO SAIA

The history of a successful marriage between theoretical mathematics and the modern computer science. Although very few people understand the axioms that regulate the use of cryptographic techniques, millions of people every day use them to safely perform their computer activities. Based on the exchange of sensitive information, these activities affect every area, from simple e-mail to sophisticated financial services

The typical *modus operandi* of the computer science community is certainly more oriented to pragmatism than to fully understanding what underlies the techniques and tools used. Specifically, the community uses a very mechanical approach, where the only requested knowledge is the final target to achieve: an obvious example of this attitude in recent years was the spread of wireless networks, where users typically do not enable any protection, simply verify that their systems worked. This kind of scenario is not always due to a mere superficiality of those who utilize this work, but rather in almost all cases, it is entirely due to the constant and exponential growth of computer industry in recent decades. This rate of change is something vastly more pronounced than what has happened in the past, and this continuous and rapid change makes a thorough understanding of the techniques and tools used in the *de facto* model extremely difficult to achieve. This article will try to fill one of these gaps by showing the close connection between the mathematics and modern cryptographic systems. Without claiming to achieve full completeness, the goal here is to expose some of the most important mathematical theories that regulate the operation of modern cryptography. We begin with the introduction of the *prime numbers*, which are the foundation for modern public-key cryptography and techniques that have become a ubiquitous standard.

Fascination with prime numbers

Prime numbers are a subject that have fascinated mathematicians for hundreds of years, inspiring novels and movies – an area so vast and complex that some areas can still be considered unexplored. For this reason what we will say here must be considered as a mere starting point for interested readers.

For a number to be defined as a *prime*, the number must be divisible only by itself and 1. More formally, a number *n* is considered prime if, and only if, it is only divisible by **±n** and **±1**. Conventionally, (the reasons will be explained later) the number 1 is not classified as a prime number, despite it perfectly fitting the *primality conditions*. These conditions are used to determine whether a number is prime, and they are true for the number **1**, because for **n=1**, we have that n is divisible only by **±n** and **±1**. Before explaining the reasons for this strange exclusion, it is appropriate to digress for a moment in order to mention that for the hundreds of years mathematicians have been trying to find a method to identify all prime numbers (assuming that a formula for this exists), and they still appear to be very far from that goal. The high performance of modern computers and parallel computing approaches were able only to isolate huge quantities of prime numbers, more or less in an empirical way without using any precise identification method. During the *prime numbers* isolation process, there began to emerge one of the mysteries that accompany primes: their distribution.

This particular branch of mathematic studies has shown how the distribution of primes are quite irregular.

The law of scarcity of prime numbers

The irregularities in the distribution of prime numbers my not be clear in the previous example, though it begins to become

more evident as the range increases. Before the number 10, there are only 4 prime numbers, which grows to 24 before reaching 100, and 168 before the number 1000, and then only 10 numbers in the first 100 integers after ten million. From this arises what in mathematics is called *law of scarcity of prime numbers* (*http://www.math.rochester.edu/about/newsletters/spring99/infinity.html*), which asserts that when the numbers examined are big, the frequency of prime numbers decreases, become increasingly rare as it tends to infinity. According to this law, we could be led to believe that after a certain number there are no more primes, and that they differ from other numbers by not being infinite. In order to solve this doubt we can refer to *Euclid*, who had proved prime numbers are infinite in a very elegant way. His demonstration type is called *reductio ad absurdum*, a Latin phrase that we can translate to *proof by contradiction* (also called indirect proof), a method used in mathematics to identify a logical process, where one initially assumes as true a certain hypothesis, a hypothesis that when advanced, leads to an absurd conclusion, a clear contradiction that indirectly demonstrates what we want. Its use dates back to ancient thinkers such as *Zeno* and *Euclid*.

The principle of the excluded middle
The *reductio ad absurdum* logical demonstration makes use of *tertium non datur* principle (the excluded middle), the principle that any statement that can not be false, must necessarily be taken as true, because there is not a third possibility; preferred by the great thinkers of the past to the method of proof by exhaustive testing, where it is necessary to consider all possible cases, it has become the favourite approach, especially in mathematic environment (*http://en.wikipedia.org/wiki/Principle_of_the_excluded_middle*).

Euclid, who so loved this kind of demonstration, proceeded in this way:

- He hypothesized that the prime numbers are finished
- Consequently, there must be a prime number **M** greater than the others
- Performing the product between **M** and the remaining numbers that precede it, and increasing by 1 the final result, we get a new prime number N greater than **M**, because when we divided **N** for each prime number we obtained as the remainder always 1
- This previous step shows an absurdity, since M was meant to be the largest prime number, and this practice demonstrates that the primes are infinite.

Here is a summary of this idea in a more formal way: *given the set of all primes **P={2, 3, ..., pn}**, where pn is the largest one of these, assuming that a N is the product of n prime numbers and consider **a+1**. This last one is not divisible by the first prime number **2**, because it is **a** (as product of all prime numbers) and so produce 1 as rest; the same situation persists for all remaining prime numbers; in a few words, assuming that $\mathbf{p}_i$ is the **i-th** prime number, the division (**a + 1)/**$\mathbf{p}_i$ always produces **1** as rest.*

In number theory, the "fundamental theorem of arithmetic" (or the unique-prime-factorization theorem) states that any integer greater than 1 can be expressed as a unique product (up to ordering of the factors) of prime numbers (http://en.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic). Based of this theorem, there are only two alternatives:

- The number **a+1** is prime and, of course, it is also larger than **pn**
- The number **a+1** is not prime and it is therefore the product of prime numbers different from those hypothesized n and then, given that **(a+1) > a** and therefore also more of pn, in this circumstance it is larger than $\mathbf{p}_n$.

Both cases, however, show a clear contradiction, since there cannot be a prime number bigger than $\mathbf{p}_n$, this demonstrates that the primes are infinite.

The factorization process

In order to formalize better what has been said previously about the process of factorization, we can assert that any integer *n* greater than *1* can be factored (reduced to factors). In other words, for each integer *n*, it is possible to locate a particular set of numbers that, when multiplied, gives as a result the number itself. Because the factorization process is not unique (we can obtain the same result with different factorizations), in order to reach a unique factorization, we must restrict the range of factors that can be used only to prime numbers. This allows us to reach the goal, since for each natural number there is only one prime factorization. Earlier, there was mentioned the conventional non-inclusion of the number *1* in the set of prime numbers. This is mandatory because if we include it, while limiting the range of factors that can be used only to prime numbers, each natural number leads to an endless factorization. That is, we always can add an arbitrary sequence of *1* to the product of prime numbers without altering the final result:

N = A x B x C x 1
N = A x B x C x 1 x 1
N = A x B x C x 1 x 1 x 1

As we can see, these above factorizations are perfect equivalents of "n", where A, B and C are the prime factors that decompose "n". In summary, in order to ensure uniqueness in the factorization, it is essential both to include only those factors that are prime numbers and exclude the number 1. In accordance with these restrictions, we see in the following example each number will have a unique factorization.

132 = 2 x 2 x 3 x 11
3900 = 2 x 2 x 3 x 5 x 5 x 13
66300 = 2 x 2 x 3 x 5 x 5 x 13 x 17

Here is a factorization notation that we can use in a more readable and compact way, employing the power notations in place of some multiplications,

132 = 22 x 31 x 111
3900 = 22 x 31 x 52 x 131
66300 = 22 x 31 x 52 x 131 x 171

There are different methods, more or less complex, to obtain the prime numbers. One of these is the well known 'Sieve of Eratosthenes' (*http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes*). It is an algorithmic process (then able to achieve the result in a finite number of steps) that finds all prime numbers below a certain predetermined maximum number **M**; the first step of algorithm consists in the preparation of a list of integers between 2 and M; follows the removal of all multiples of 2 except 2 itself; the next step starts from the first number not removed (then 3), removing from the list all its multiples, always excluding 3 itself; continuing in this way until reaching the integer part of the square root of M, at the

end of the operations, there will remain only the prime numbers in the range from 2 to M.

In the example, assuming **M=25**, the integer part of its square root is **5**: proceeding with the method described above, applied to all numbers up to 5, we obtain the prime numbers in the range from **2** to **25**. Again, in accordance with previously mentioned precedent considerations, the number **1** was not included.

Factorization and asymmetric encryption

The practical application in cryptography tied to what we have just seen, stems from the fact that for pretty big numbers it is very difficult to identify the prime factors that comprise its factorization. It is so difficult that, beyond a certain numerical value, techniques and tools available today are inadequate, due to excessive time required for the operation. Today in fact, there are no methods capable of performing this type of operation within a reasonable time. For example, in order to identify the factors of a 100 digit integer, we indicatively need 1 second, 1 year and half for a 200 digit integer and 2 million years for a 300 digit integer (*http://eprint.iacr.org/2010/006.pdf*). For this reason, and in consideration of today's asymmetric encryption algorithms that typically use a public key of at least 1024 bits, (the equivalent of about 300 digits) it is easy to understand how the modern encryption methods are inviolable in practice. Several years ago, almost all of the encryption algorithms in use were symmetrical, based on a single encryption key, a key absolutely essential for the encryption and decryption. The vulnerability of this kind of system is clearly represented from this key, because it must circulate among all authorized users, with clear implications for security. The real revolution in this area was the adoption of a completely different mechanism from that formerly used. A new approach was developed that uses an asymmetrical method, which has different keys for encoding and decoding of data. This is a simple but incredibly efficient method that has revolutionized the cryptography environments. What happens here is that information is encrypted with a public key, but then the resulting encrypted parcel can only be decrypted using a complimentary private key, a key possessed only by the legitimate recipient of information. Asymmetric cryptography is also called *public key cryptography*, as opposed to the symmetric cryptography, which is instead defined 'private key cryptography'. A common element in any encryption mechanism, symmetrical or asymmetrical, is the effectiveness of the key that is used (public or private). A simple key can invalidate the potential robustness of asymmetric mechanism.

Keys with inadequate complexity (length, used characters, etc.) can be deciphered through trivial brute-force techniques that are repeated attempts based on several dictionaries or on lists that contain the words most used by users. The operation of asymmetric algorithms is based on two related primary elements, the public and private keys. The mechanism used to generate these keys does not permit in any way a means to obtain the private key from the public key, this is connected with the enormous difficulty of individuate the prime factors that determine a certain number.

Some well-known algorithms of this type include RSA (an acronym derived from the first letters of developer surnames, Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman) and DSS (Digital Signature Standard). To understand how these algorithms work, we will assume the existence of two systems (Alpha and Beta) that wish to exchange information securely. Alpha and Beta have a pair of keys (one public and one private), Alpha asks Beta for its public key and uses that to encrypt the data that is sent to Beta. This data encrypted with Beta's public key can be decrypted only with Beta's corresponding private key. Beta uses its private key and decrypts the data sent by Alfa, the same process is used in the other direction. The private key is the only key that remains secret during the entire process, because the public key can be disclosed without problems. It is useful to underline that in practice the first step (request of the public key) does not occur, since the request is usually automatically managed by some servers where users public keys are stored. Years ago, the RSA algorithm was tested for robustness. To do this, a consortium including about 1600 computers and the efforts of 600 study groups located in more than 25 different countries tried to violate a 129 bit-length key. The violation was ultimately successful after more than 8 months of uninterrupted work of all computers (*http://en.wikipedia.org/wiki/RSA-129#RSA-129*). This experiment has shown the practical inviolability of this cryptographic algorithm, in consideration that the modern keys are longer than 129 bits: we just think that a modern protocol such as AES (Advanced Encryption Standard) uses a 256-bit key.

A theoretically unbreakable system

We can observe as the inviolability of today's public-key encryption systems is only a "practical state" caused by the inefficiency of current factorization methods of the big prime numbers and is thus a scenario that could change in future, invalidating completely the algorithms currently used (*http://www.dtc.umn.edu/~odlyzko/doc/future.of.factoring.pdf*).

The Vernam cipher

Regarding the current non-availability of an unbreakable encryption system, it must be said that this statement can be considered partially false because it has been mathematically proven that an unbreakable encryption system exists. The *Vernam cipher*, also known as *ideal cipher* is the only encryption mechanism that can be defined as *mathematically certain*. A mechanism of formal proofs exists to demonstrate its inviolability since the end of the 1940's. Briefly summarized, the *Vernam cipher* had its origins in 1918, the year in which Gilbert Vernam, a technician at the AT & T Bell company and Officer in the U.S. Army, improved the encryption method called *Vigenere method* (one of the simplest existing ciphers based on the use of a phrase to control the alternation of the alphabets of substitution); Vernam extended this cipher with random encryption keys of the same size as the message to be encrypted, creating what was later called the *Vernam cipher*.

In 1949, Claude Shannon, the undisputed father of Information Theory, published a paper entitled *Communication theory of secrecy systems*, document which shown in a rigorous way as the *Vernam cipher* can be considered a (mathematically) secure encryption mechanism.

Infinite key encryption

The apparent inviolability of the *Vernam cipher* hides a problem, because the encryption key can be used only once. For this reason the *Vernam cipher* is included in the category of the cryptographic systems that use non-reusable keys, systems that are classified as *with infinite key encryption* or *One Time Pad*. While taking into consideration this complication, it is logical to wonder why, assuming that there is a secure encryption system, it is not used. The answer to this question is disarmingly simple and is based on three considerations:

- The required encryption key is as long as the message;
- This key must be sent to recipient;
- The key transmission requires a secure channel.

In light of these limitations, it is logical to ask why, instead of using the secure channel to exchange the key, we don't use this channel to send the message directly? This is a clear example of how some solutions are only theoretically valid.

Diffie-Hellman vs Man-in-the-middle

Regardless of the type of cryptographic system, the weak point is always the exchange of the keys. This is an operation that leads to a potential risk of decoding by the attacker or, in the worst case, when there is not an adequate key complexity, to a total compromise of the information. In this regard it is appropriate to introduce the *Diffie-Hellman Protocol.* This is a protocol publicly formalized in the mid-seventies, and as its name suggests, was the result of collaboration between Whitfield Diffie and Martin Hellman. It allows two users to securely reach a common encryption key using a regular unprotected communication channel. This key is used in symmetric cryptographic systems.

Known as the *Diffie-Hellman key exchange*, this protocol is based on prime numbers and assuming that the roles are again Alpha and Beta, the operations performed are as follows:

- Alpha and Beta agree to use a prime number **p=19** and a base **g=2**;
- Alfa chooses a secret number **a=5** and sends to Beta **A = ga mod p**, that is **13**;
- Beta selects a secret integer **b=10** and sent to **Alpha B = gb mod p**, that is **17**;
- Alpha calculates **KEYA = (gb mod p)a mod p,** that is **6**;
- Beta calculates **KEYB = (ga mod p)b mod p**, that is **6**.

The term *mod* identifies a *module operation*, that is an operation that returns the remainder of a division (for example, **5 mod 3** produces as result **2**). As you can see, at the end of protocol procedure Alpha and Beta obtain the same result (**KEYA** and **KEYB**), only elements that in combination with *a* and *b* remain secret. That is, the remaining values are not encrypted. The common result **KEYA** and **KEYB** represent the secret keys. It should be emphasized that the initial values in the real applications must be far greater than those used in the example, except for the g value that usually is a value of **2** or **5**. We can say that when a and b are numbers of at least one hundred digits and p is a prime number of at least 300 digits, the searching of the secret number (when are only known values **g**, **p** and **ga mod p**) is impractical, even using the resources of all the computers that exist today.

The reason for this inviolability leads back to the concept of *discrete logarithm* (*http://en.wikipedia.org/wiki/Discrete_logarithm*), an operation that has an order of complexity similar to the *integer factorization* (*http://en.wikipedia.org/wiki/Integer_factorization*).

Discrete logarithm

The logarithm is a mathematical operation opposed to the exponentiation and, in parallel, the *discrete logarithm* expresses the inverse operation to *discrete exponentiation*, suppose to have a set N containing integers from 0 to m-1, where m is a prime number, we can define with a(PD)b = ab mod p the *discrete exponentiation* operation of two numbers a, b, both belonging to the set N, and hence defined as *discrete logarithm* of a number x based a the number b for which a(PD)b = x, that is ab mod p = x.

In this case, although for different reasons, the theoretical inviolability collides with practical implementation, as this procedure proves to be highly vulnerable to an attack carried out according with technique of the *man in the middle* (*http://en.wikipedia.org/wiki/Man-in-the-middle_attack*), where the attacker interposes itself between the parts in order to change the original public key with a suitably forged one: that is possible because users initially must exchange this kind of information.

Quantum cryptography

One of the most promising areas of science for the security of communications, is surely *quantum cryptography* (*http://en.wikipedia.org/wiki/Quantum_cryptography*), an ambitious project that aims to combine some aspects of quantum mechanics with the cryptography.

Advantages of Quantum Cryptography

The main benefits of the Quantum Cryptography are the ability to: provide absolutely secured keys, detect intruders on optical networks; allow high-speed key updating; simplify the key management, providing timeless security and removing the human risk within the security chain

For this reason, in addition to the interest of the usual operators, the high security that this technology provides has aroused interest in many other environments, such as military and government. One of the first practical applications of quantum cryptography is called QKD, Quantum Key Distribution (*http://en.wikipedia.org/wiki/Quantum_key_distribution*). It aims to solve the longstanding problem of securely distributing cryptographic keys through a conventional (not secure) transmission channel.

This technique takes advantage of some properties of elementary particles and of the laws of quantum physics in order to ensure a safe exchange of information. This allows you to operate safely within a symmetric cryptographic environment: although the symmetric encryption is much faster than asymmetric, the latter is preferred because it does not require a preliminary exchange of encryption key (shared by users), high-risk activities without a secure channel such as, for example, a quantum channel. That is, an environment with very high performance, that is seldom used presently because of the necessary encryption key exchange risks.

BB84 Algorithm

Another example of the "Quantum Key Distribution" refers to the implementation of the algorithm called BB84, an algorithm formalized in the first mid-eighties by Charles Bennett and Gilles Brassard. It is the first quantum cryptography protocol

To exploit the possibilities offered by QKD it is essential that the partners share a quantum communication channel that can be simplex (unidirectional) or duplex (bidirectional). They must also have the appropriate tools to send and receive photon particles. Assuming two rightful users Alpha and Beta and an attacker Gamma, and assuming the existence of an optical fiber transmission channel (the quantum communication channel) capable of transmitting single photons (neutral particles characterized by a velocity of propagation in vacuum of about 300,000 km/s, in practice, they are photons generated by a laser light), the operations timeline is summarized below:

- Alpha sends single photons to Beta through the optical fiber.

These photons are randomly oriented by Alfa based on a set of four possible angles (0, 45, 90 or 135 degrees)
- Beta will verify the polarization of the photons received by using a filter able to identify the different angles; Beta will choose (arbitrarily) a reference base of 0 and 90 degrees or 45 and 135 degrees
- Beta, using an ordinary communication channel, will inform Alfa about the method used but not about the result obtained
- Alpha will inform Beta about which orientations are correct for him, allowing Beta to delete all photons flagged as incorrect and convert the remaining ones in binary format (1 or 0) on the basis of their orientation; these last values (once joined) will represent the encryption key to be used;
- If Gamma intercepts the communication, he can determine the direction of the photons, but his measurement can not be confirmed by anyone. Consequently, it will be impossible for Gamma to interpret the key in the correct way.

The process just described is briefly (where the numbers beside the orientation are the binary values to be attributed to the particle), that concretized a typical *one-time pad* scenario. It should be mentioned that the legitimate interlocutors (Alpha and Beta) are not able to know in advance which key will be used, because it depends on their arbitrary choices. Another aspect that is useful concerns the invulnerability of such a system. In this case security is not derived from the impossibility for an attacker to intercept communications but from the uselessness of this captured transmission.

The Hilbert Space

Just a brief mathematics aside in relation to the reference orientations chosen by Beta. Because they configure a base of the *Hilbert space*, a generalization through the "vector space" of the *Euclidean space* introduced by the mathematician David Hilbert at the beginning of the twentieth century. A generalization worthy of note, considering the crucial role played in the quantum mechanics (*http://en.wikipedia.org/wiki/Hilbert_Space*).

The Heisenberg Uncertainty Principle

We can not finish this article without mentioning one of the pillars of quantum physics as applied to cryptography – *the Heisenberg Uncertainty Principle*. It was published in the second half of the twenties by Werner Karl Heisenberg, a Nobel Prize winner in physics and founder of the quantum mechanics. Explained in the simplest possible way, this principle states that it is not possible to know simultaneously two aspects of quantum properties of elementary particles (for example, in the case of a photon, its velocity and position). Although many distinguished scientists, including Albert Einstein, have attempted to refute this principle: during his life he did not want to accept it, he was not disposed to admit, even in principle, the impossibility to discover all the facts necessary to describe a physical process, attributing the inability to perform this operation to the limits of the techniques and instruments used for measuring. In the following years it was proved that such limitations actually exist, regardless of the effectiveness of techniques and measuring instruments. The *Heisenberg Uncertainty Principle* directly depends on a property of fundamental importance for the security of information transmitted by quantum cryptography systems. Any attempt to intercept or detect some properties of the particles that make up the information will inevitably lead to a permanent change in their quantum state. This will also make it impossible to perform further measurements. In this way it is clear that the legitimate interlocutors are able to detect when a communication has been intercepted. With regard to the BB84 protocol, the assurance that legitimate communication between the intended parties has not been intercepted by third parties (in the case of the example from Gamma) derives from the observation that if Gamma had intercepted photons transmitted between Alpha and Beta, according with the *Heisenberg Uncertainty Principle*, these would have been inevitably altered, thus introducing some errors in measurements of Beta and consequently in the binary values (the key). Following such an instance, Beta will eliminate all photons reported as incorrect by Alfa, converting the remaining ones to binary values to be used as the encryption key, Beta will generate a different key from the Alpha one and this difference indicates unequivocally the interception by Gamma.

BB84 protocol hacking

What has been said so far could lead the reader to think that the BB84 protocol is today the ultimate solution in the field of cryptography, and this is partly true – regarding the theoretical implications. One of the problems about its practical applications derives from the current inability to completely shield the communication systems in order to make them immune to disturbance. The result is a systematic percentage of errors affecting systems (now estimated at around twenty percent). Exploiting this aspect and taking advantage of the fault tolerance that these systems must have, a quantum physicist Hoi-Kwong Lo and his colleagues at the University of Toronto in Ontario, Canada, have hacked a cryptographic system based on *Quantum Key Distribution* (then the BB84 protocol) located in Geneva, Switzerland (*http://en.wikipedia.org/wiki/Quantum_key_distribution#Photon_number_splitting_attack*). In relation to the operational timeline mentioned above, they broke the system with the photons orientation process (that is when Alpha sends photons to Beta) in order to make the intercepted key valid. The system was 'confused' by introducing artificially a high percentage of errors, errors completely indistinguishable from those endemic to the system.

Conclusions

In this article we have shown how the modern cryptographic techniques are continually improving the security of communication systems. The future seems to offer amazing applications of the laws of quantum mechanics. Applications oriented to the concrete realization of *quantum computers* (*http://en.wikipedia.org/wiki/Quantum_computer*), systems capable to operate with an absolutely revolutionary logic, which is able to perform tasks deemed impossible today, as the execution of large computations in a few moments. When quantum computers will be a reality we instantly could obtain a private key from only the public key, therein completely compromising the security of any public key cryptographic algorithms currently used. Before that day, we will have to radically change the way we do encryption, reviewing everything according to the new technological opportunities. Opportunities that might deliver the dream of cryptographers to create a system that is unbreakable.

ROBERTO SAIA

Graduated in Computer Science, Roberto Saia professionally works in the ICT sector; for several years he has been managing computers network and security of a large national company; author of numerous books on programming, administration and system/network security, for some time his interest is mainly focuses to the security environment, in the broadest sense of this term.

Easily add security at the source

buguroo
offensive security

In buguroo our **expert teams in security, hacking and programming** allows us to find solutions to simplify the development of secure code to our customers.

"Simplicity is the ultimate sophistication"
Leonardo da Vinci

buguroo has designed and developed bugScout, a powerful managed service for **source code vulnerability analysis**:

- **Effectiveness.** bugScout automatically detects over 94% of the vulnerabilities that can be found within the source code.
- **Simplicity.** bugScout includes a project, application and analysis classification system, incorporates a reports manager and makes vulnerability management a lot easier.
- **Scalability.** bugScout works in a decentralized, cloud computing environment.
- **Parallelized.** bugScout is designed to simultaneously audit multiple source codes without affecting performance.
- **Customizable.** bugScout is a multitasking and multiuser platform providing for rights granularity. The user interfaces are completely customizable.

SSL/TLS PKI ISSUES

MARTIN RUBLIK

The most common network protocol that is used for encrypting web communications is SSL/TLS. During the last year we experienced several security incidents and this showed us that the risks associated with the management of cryptographic keys (not only in SSL/TLS) are real and relevant.

The motives for protecting the network communications by encrypting it are straightforward. It is quite easy to eavesdrop on network communications especially on wireless networks, or networks in a hostile environment controlled by an attacker. By eavesdropping on network communication an attacker can gain access to vital information. We use the communication networks for shopping, work and fun and we send quite a lot of sensitive information through them. If an attacker is able to access unencrypted/unprotected/plain communications, they can very easily impersonate the victim and gain access to victim's digital identity and accounts.

Researchers have come up with several means of protecting network communications using encryption. The purpose of encrypting the network communications is to preserve its attributes of confidentiality, availability and integrity. The most common network protocol for protecting the internet communications is SSL/TLS. SSL/TLS is used for protecting the web communications (HTTP/SSL), email communications (SMTP/SSL, IMAP/SSL, POP3/SSL) and also for protecting other kind of network communications like LDAP, FTP, RDP, XMPP/Jabber.

SSL/TLS protocol cryptography basics

The SSL/TLS protocol uses three kinds of cryptographic primitives to protect network communications. It uses symmetric encryption algorithms for protecting the confidentiality of communications, message authentication codes for protecting integrity and authenticity of the communications and asymmetric encryption algorithms for key exchange.

In symmetric encryption the sender converts/encrypts data/plaintext to encrypted data/ciphertext that is unreadable for an attacker. The recipient of the encrypted data decrypts the data to its original form and is able to read the data. To preserve the confidentiality of the data there must be some sort of information that is not known to the attacker. It can be either the algorithm that is used for encryption/decryption or an additional parameter that is passed to this algorithm. The encryption/decryption can be easily reverse engineered and thus does not constitute adequate protection, one needs an additional parameter that is called cryptographic key. This key needs to be safely transferred between the communicating parties so that the attacker cannot decrypt the information being transmitted.

Symmetric encryption is mostly used to protect the confidentiality of data. Though some symmetric encryption algorithms can be used also for protecting integrity and authenticity of data, in general this is not true. Therefore, if a symmetric encryption algorithm that does not protect the integrity and authenticity of data is used, one needs to use an additional cryptographic protection mechanism. Message authentication codes (MAC) can be used for this purpose in SSL/TLS network protocol. Basically MAC is a product of special one way cryptographic function (either based on symmetric encryption algorithm or hash function). This output can be used for verification of integrity of input data. Both communicating parties (creator of the MAC and its verifier) need to agree on the same cryptographic key that is used for creating and verifying a MAC.

Asymmetric encryption algorithms can be used for protecting confidentiality of data. Asymmetric encryption algorithms are based on different principles than symmetric encryption algorithms. The asymmetric encryption algorithm needs two cryptographic keys that are mathematically tied together. First key is used for encrypting the data and it can be available to everyone, especially sender of the data. This key is called a public key. Second key is called private key and is only available to the recipient of the data. This key is used for decrypting the data by the receiver.

The benefits that asymmetric encryption algorithms have over the symmetric encryption algorithms reside especially in simplified key distribution. It is not necessary to protect the confidentiality of public keys that are used for encryption of data. Unfortunately, asymmetric encryption algorithms are much more computationally demanding than symmetric encryption algorithms and are therefore used mostly for encrypting small amounts of data. One of such use cases is the symmetric key distribution. In this particular use case the symmetric encryption algorithm is used for protecting the actual data (making the encryption faster) and asymmetric encryption algorithm is used for exchange of symmetric keys (making the key distribution easier).

Though asymmetric encryption makes key management and distribution much easier, there are still several problems that need to be solved:

- integrity and authenticity of cryptographic keys,
- bonding between the cryptographic keys and their holder,
- validity and revocation of cryptographic keys.

These problems can be solved by several different approaches. First approach is out of band key distribution through a secure channel. This approach needs another secure channel in order to

distribute a key (for example through personal interaction, USB, etc.) and can be used in circumstances where one communication party knows the other.

The second approach is accepting the public cryptographic key for the first time one party sees it (possibly verifying the integrity by using the band channel, like telephone or other means of personal interaction) and storing it for future use. This type of key exchange is used for example in SSH protocol.

The trouble with first two approaches is that they really do not scale well for large communities (e.g. web shops like Amazon) and it is quite hard to perform revocation of cryptographic keys. Amazon does not really want to prepare a call-center just for verifying the integrity and authenticity of cryptographic keys. Also this approach is not comfortable for an average user.

The third approach is based on a premise that there exists a trusted third party (TTP) that performs key certification. This TTP performs identity validation, and issues a statement where it bonds the public key and the identity of its holder. The statement is protected by cryptographic means with the key of the TTP. This way the problem of key distribution is reduced to distribution of one/several TTP cryptographic keys and providing a public key infrastructure (PKI) that solves the bonding between cryptographic keys and the identity of their holder as well as dealing with revocation of cryptographic keys. SSL/TLS protocol, how it is in use today, utilizes the third approach.

SSL/TLS PKI basics

SSL/TLS PKI is based upon X.509 public key certificates. In X.509 public key infrastructure the trusted third party is called a certification authority (CA). CAs issues the certificates in order to bind the holder's public key to a domain DNS name that is used for network communications.

The certificate has structure as defined in "ITU-T X.509 Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks". The general structure of certificate is illustrated in 1. A sample certificate can be viewed in browser by clicking the address bar padlock after SSL/TLS channel establishment.

The CA can either issue certificates to other CA's or directly for the users (also called end entities). The reason for having a CA hierarchy is to simplify the CA's key/certificate distribution. This way it is possible to build an infrastructure where there is only small amount of root CAs that issues certificates to several subordinate CAs and you need to distribute only the small number of the root CA certificates.

Every CA can have its own scope of issuing certificates and own set of policies that place requirements on operational procedures, security precautions and procedures for end entity identity verifications.

In TLS/SSL PKI there are three most common types of identity verification:

- domain validated,
- organization validated,
- extended validation.

The content of the public key certificate is based on identity validation process. Domain validated certificates contain in subject name (holder information) only the DNS domain name that is owned by the end entity (e.g. CN=*www.domain.org*). These certificates are easiest to obtain and are cheapest. The validation process is/can be done in an automatized way and involves mostly WHOIS checks or sending verification e-mails to a pre-configured e-mail address (such as *hostmaster@domain.org* or *postmaster@domain.org etc.*). The e-mails typically contain a one-time password that is used for identity validation during the certificate issuance process.

Organization validated (OV) certificates contain more information about the end entity. Besides the checks concerning the ownership of a domain, there are also other types of checks concerning vetting the organization. These checks can be based on public registers of organizations or based on scanned invoices. The process that is used for validation of OV certificates is semi-automated and could involve a personal contact between the certification authority officer and the end entity. The OV issued certificate contains in subject name the DNS domain that was checked as well as information about the organization that is the holder of domain (e.g. the subject could look like CN=*www.domain.org*, O=*Organization Inc.*, L=*Wien*, C=*AT*).

The third type of identity verification process is extended validation (EV). The EV issued certificates have similar verification process to OV certificates, and in addition, a more thorough/rigorous verification process of applicants. Furthermore, the process is standardized by CA browser forum. Some other technical requirements imposed on the certificate are the need for stronger cryptographic algorithms1 and stronger cryptographic keys2. Also EV protected network communications are indicated by internet browsers using a green address bar.

The third type of identity verification process is extended validation (EV). The EV issued certificates have similar verification process to OV certificates, and in addition, a more thorough/rigorous verification process of applicants. Furthermore, the process is standardized by CA browser forum. Some other technical requirements imposed on the certificate are the need for stronger cryptographic algorithms and stronger cryptographic keys2. Also EV protected network communications are indicated by internet browsers using a green address bar.

If the web site administrator wants to enable SSL/TLS protection of the network communication he needs to obtain an X.509 certificate from certification authority. There are several commercial certification authorities that issue X.509 certificates. The web site administrator should choose a CA that has its certificate shipped with major internet browsers. This way the user won't get any warning messages during the SSL/TLS channel setup and the network communication is less prone to eavesdropping.

If the web site administrator obtains a certificate from a CA that is not shipped with internet browser, the browser issues a warning to the user. The user should verify the authenticity of the certificate afterwards. However this is seldom done by users and it is really inconvenient. If the certificate is not verified by the user, an attacker could perform a man-in-the-middle attack where he would create a certificate with same name but issued by a CA that he can control.

After the web site administrator configures the web server, it can start to serve content over SSL/TLS protected channel.

How to attack SSL/TLS encrypted channel

There are several ways to attack an encrypted channel. The level of complexity of an attack depends on the level of sophistication of the process/IT security of its target. The attacker can choose to target the encryption algorithms (either from mathematical point of view, or from implementation's point of view), or attacker can choose to target the protocol (again either from design point of view or from implementation point of view), or an attacker can choose to target the key management and distribution infrastructure.

Though there are known attacks on encryption algorithms used in SSL/TLS protocol, these attacks are too far from being practi-

cal. These attacks are big topics for mathematicians but for internet users, system administrators and attackers as well, they are absolutely irrelevant.

There is a known attack against TLS 1.0 protocol implemented by Rizzo and Duong (BEAST attack), but it is still not easy to successfully execute this attack. The attack requires the user's interaction (e.g. a visit to a special crafted malicious web site), it also has some requriements on communication's behavior (e.g. the communication needs to have a secret, for example a cookie, on predictable location).

On the other hand, in the previous year alone, there were at least two known compromises to SSL/TLS public key infrastructure. As a result of these attacks Comodo CA and DigiNotar issued fradulent certificates to an attacker that did not control the specific DNS domains. This way attacker could mount a successful attack against any web site that uses SSL/TLS protocol for protection of network communications as long as he could control the network on lower layers. There are some indications that these fradulent certificates were used during an Iranian attack on public e-mail services such as Google Mail.

The attack is quite simple. During the first phase of SSL/TLS protocol the attacker performs a man in the middle attack by inserting a legitimate certificate from compromised CA. If the CA is not aware of being compromised and/or the attacker can block CRL/OCSP communication with CA, the browser cannot determine whether the certificate was fradulently issued and consequently, considers the certificate as valid. Afterwards the client will encrypt the symmetric key with attacker's public key and attacker can decrypt all the traffic designated for the web server. He can than act as an active proxy server in the communication.

There are several reasons why this attack is the easiest one. Their summary is as follows:

- revocation either by CRL or OCSP is by default fail-safe (meaning that if the browser is unable to retrieve revocation information the SSL/TLS channel setup proceeds, and sometimes it proceeds even without a warning),
- the users are not educated enough, meaning that the attacker does not even need to attack the CA. He can issue a certificate with a non-trusted CA and many users will just ignore the browser's warning,
- the web servers are misconfigured for SSL/TLS (either by serving non trusted certificate, or by serving a certificate with mismatched name, or by serving an expired certificate, or by using deprecated cryptographic algorithms, or by violating the SSL/TLS specifications),
- there are many (perhaps too many CAs) trusted by the internet browsers.

In 2010 the researchers from EFF presented results from their SSL Observatory project. The main objective of this project was to create a view on the SSL/TLS PKI by mapping publicly available web servers that serve content through HTTP over SSL. The results are very interesting. According to researchers there are more than 650 subordinate certification authorities that are trusted by major internet browsers. This number is biased as the root CAs are not obligated to publish the number of subordinate CAs to which they have issued a certificate. Therefore the researchers decided to estimate the number based on different name in CA certificate, especially different organization value. Another estimate, which can

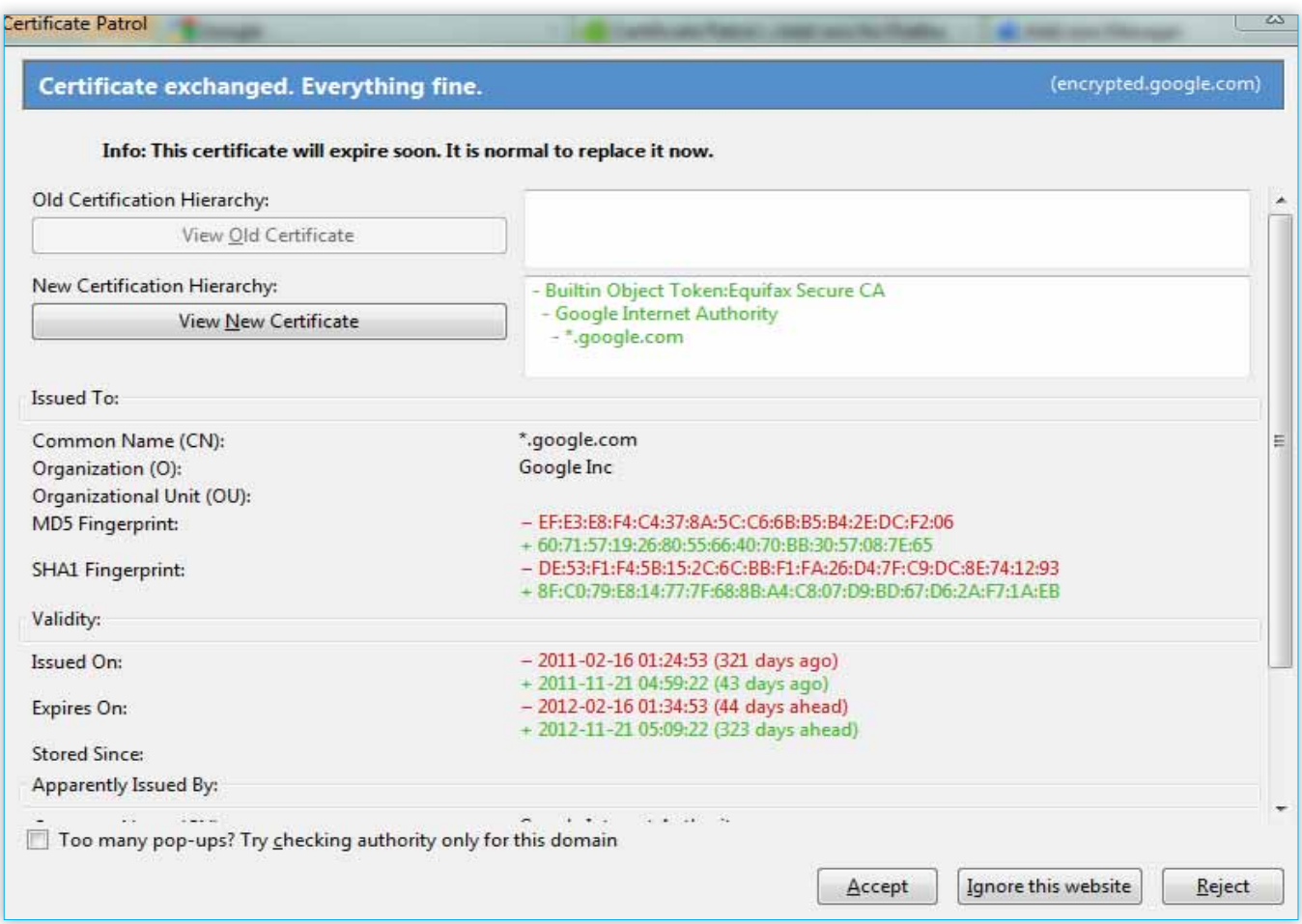


Figure 1. *Certificate patrol information about certificate change*

be considered a lower bound, is 50 CAs (this estimate considers several CAs running at same location and operated by same personnel as one CA), but unfortunately the latter estimate does not have any proof whatsoever.

Also the Observatory project showed that CA practices are not followed rigorously. For example there were found several EV certificates that were issued to a non-qualified domain name (e.g. *webmail.domain.local* or *intranet*) or certificate issued to a short key (512 bit RSA).

Another research conducted by researchers from Technische Universität München also supports the conclusion that the SSL/TLS PKI is in sorry state. Their research was more thorough than EFF's and ran in longer time span, but concluded to similar results as the Observatory project.

Present and future countermeasures

The ugly part about the SSL/TLS PKI is that the attacker only needs to circumvent the safeguards of a *single* trusted certification authority and he is able to attack almost1 any internet site that uses SSL/TLS for protecting its communications.

With such high number of CAs (even lower bound is still high enough) the attack surface is attractive to attackers. It is hard to withdraw the certification authority from the browser trust store as it would break the established trust and it would mean that all the certificates that were issued through this CA needs to be replaced. Based on current sorry state of SSL/TLS PKI mis-configurations, one can assume that many server certificates would be omitted leading to user confusion and leaving the doors open to the attackers.

There is a lot of work to be done by browser vendors and CAs. Browser vendors should update their CA inclusion policies and should require that CA publish information regarding audit reports, security practices of root CAs and issue subordinate CAs as well and breach disclosure policies.

There is also a second possibility how to improve the security of SSL/TLS PKI. This possibility is based on a cross check of CA issued public key certificates.

The most simple way to perform a cross check is to use an SSH like key continuity principle in conjunction with traditional X.509 PKI. There is a Firefox extension called Certificate Patrol that supports this kind of protection. First time a user visits a web site he can verify the SSL/TLS certificate manually and Certificate Patrol stores it locally. If the Certificate Patrol discovers that a new certificate is being used, it will show the differences and alerts the user so he can make a decision whether he trusts the new certificate.

We have discussed this approach earlier, and also identified several problems (especially that it is not very comfortable for non tech-savvy users). Besides these issues there is one more problem. If the web site uses multiple certificates (for example because of load balancers), it will issue false positives.

Another approach was proposed by researchers from Carnegie Mellon University. Their solution (Perspectives project) is based on a separate public key infrastructure and a geographically distributed network of certificate notaries. These notaries scan internet sites for SSL/TLS servers and store the certificates they see. When the browser sets-up a SSL/TLS connection it'll contact several notary servers and compares the results. The Perspectives browser plugin can be configured on how many notaries need to see the certificate consistently and for how long.

This way the attacker would need to attack the network closest to the server and for a longer timeframe, so that every notary server views the fraudulent certificate over a longer period. This makes attacks significantly harder, even impossible, especially for

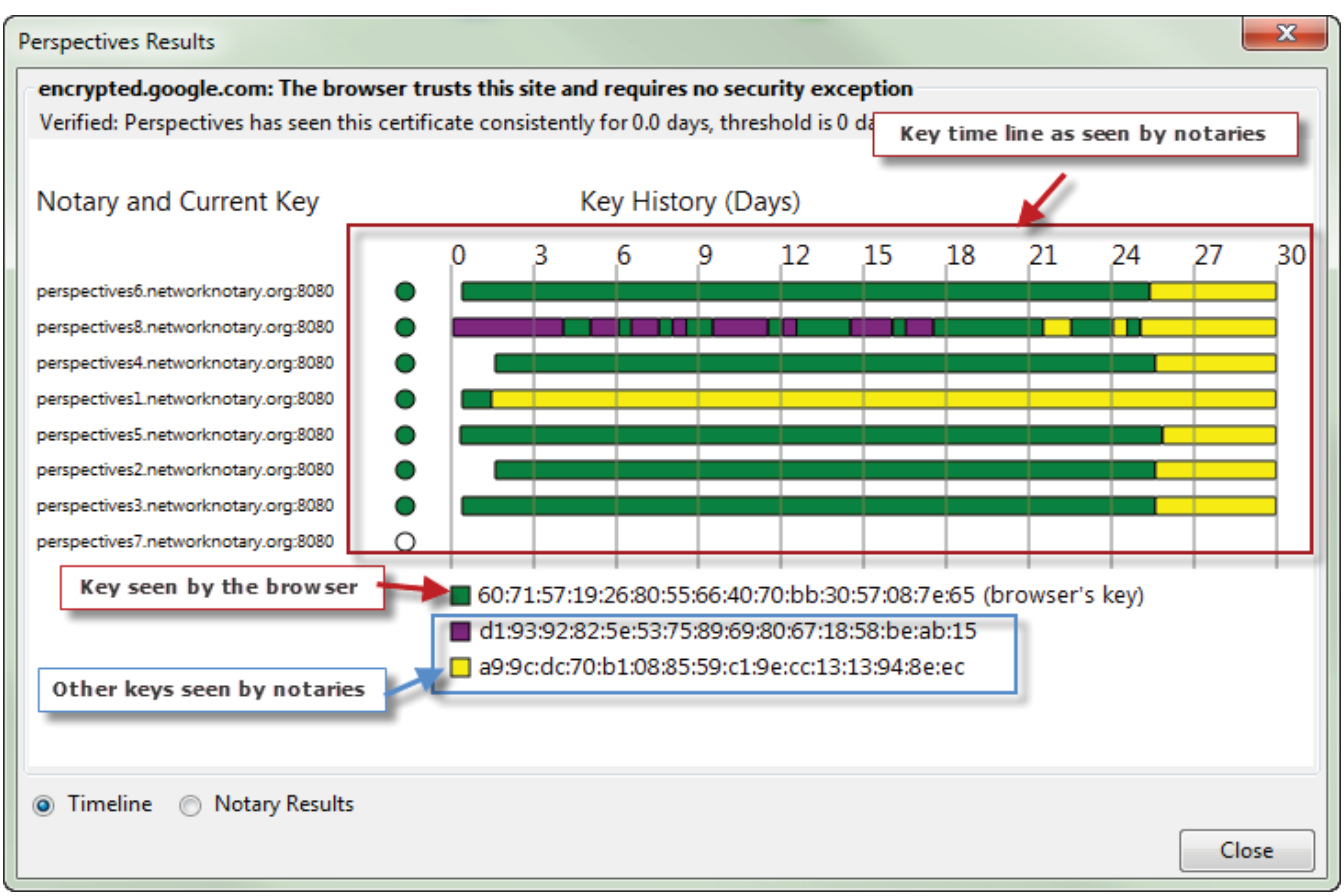


Figure 2. *Perspectives plugin certificate report*

man in the middle attack scenarios where attacker controls only the local area network.

This solution does not have the drawbacks of the key continuity, as it is more user friendly and can deal with the key rotation/ multiple certificates problem (as illustrated by 4). It even deals with the problem that can be caused when a server certificate is issued by a non-trusted certification authority, because the verification is done independently from browser's set of trusted root certification authorities.

Moxie's Convergence project is also built upon Perspectives idea. This project is currently still in beta version and it supports the same logic for determining whether the certificate is considered valid or not. However in the future it plans to rely on multiple sources of information. These could include online notaries, EFF Observatory database, CA sources and DNSSEC based information.

Besides already presented solutions there is ongoing discussion on standards track. IETF PKIX working group and IETF DANE working group are preparing standards proposals that would help to mitigate the risks connected to SSL/TLS PKI.

DANE (DNS Based Authentication of Named Entities) provides bindings between the domain name and the server's public keys / certificates. The idea behind is to use DNS protocol to bind DNS name, port and cryptographic keys. DANE extends the DNS protocol by adding a new resource record type (TLSA) to the existing resource record types (HOST, PTR, CNAME, MX, etc.). The TLSA RR will bear several types of information.

First type of information specifies which of the certificates received by the client during SSL/TLS channel setup should be used during cross-check against TLSA RR. It can be either a CA certificate (either a root CA certificate or one of the subordinate CA certificates), or it can be directly server's certificate. This information determines how the information from TLSA RR should be used by a client.

Second type of information specifies the content of the TLSA record. This can be either information regarding public key only or it can be full public key certificate information.

Third type of information specifies how TLSA association is presented. This can be either:

- full information (e.g. public key information or X.509 certificate),
- hashed information – digital cryptographic thumbprint of full information.

An example of a TLSA record according to DANE protocol draft#13 is illustrated by 5.

The DANE strengths are similar to Perspectives. It can provide cross-check to CA issued certificates and it can also be used to trust certificates directly (without the necessity of including a CA in a browser trust list). It is also worthy to mention that DANE could be easily implemented in a non-browser environment that cannot rely on user decisions and interaction (e.g. MTA software for STMP over SSL/TLS).

Unfortunately DNS is not a secure protocol and therefore one needs to use DNSSEC, in order to provide protection for associations built by DANE. However DNSSEC also relies on a public key infrastructure where the trusted third parties are domain registrars. The improvement that comes with DNSSEC is that in case of one TTP compromise, only the part of the infrastructure for which the TTP was authoritative is in danger.

Similar approach was chosen by PKIX IETF working group. The PKIX WG proposed to extend DNS resource records with CAA (Certification Authority Authorization) RR. This resource record would be used by certification authorities as part of background checks prior issuing a certificate. This way the risk of unintended certificate issuance would be reduced. The resource record should contain information about a certification authority that is allowed to issue a certificate for a specific domain, as well as contact information that would be used by CA in case when CA is unauthorized to issue a certificate (based on CAA RR). This way the domain owner can be notified that someone is trying to acquire a certificate from another CA than the one that is indicated in CAA RR.

Conclusion

The main purpose of this article was to describe the basics of SSL/TLS usage, its problems with regards to cryptographic key management and outline possible solutions. There is no easy way to solve the problem of cryptographic key management and we will see which solution will be accepted by the industry and internet users.

Besides the existing solution proposals new ones are emerging, especially EFF Sovereign Keys and Ben Laurie's and Adam Langley's work on Auditable CAs are worth mentioning. We shall follow the situation and bring you more information on the topic. In the mean time we have informed you about existing detection mechanisms (Certificate Patrol, Perspectives, Convergence) so that you can browse the internet in a safer way.

MARTIN RUBLÍK, PH.D., CISSP

Martin Rublík is a senior security consultant at BSP Consulting and a lecturer at University of Economics in Bratislava. He received his master's degree at Faculty of Mathematics, Physics and Informatics (Comenius University in Bratislava) in 2005 and his Ph.D. degree at Faculty of Economic Informatics (University of Economics in Bratislava) in 2010. His main area of expertise includes identity management, PKI, smart cards and network security.

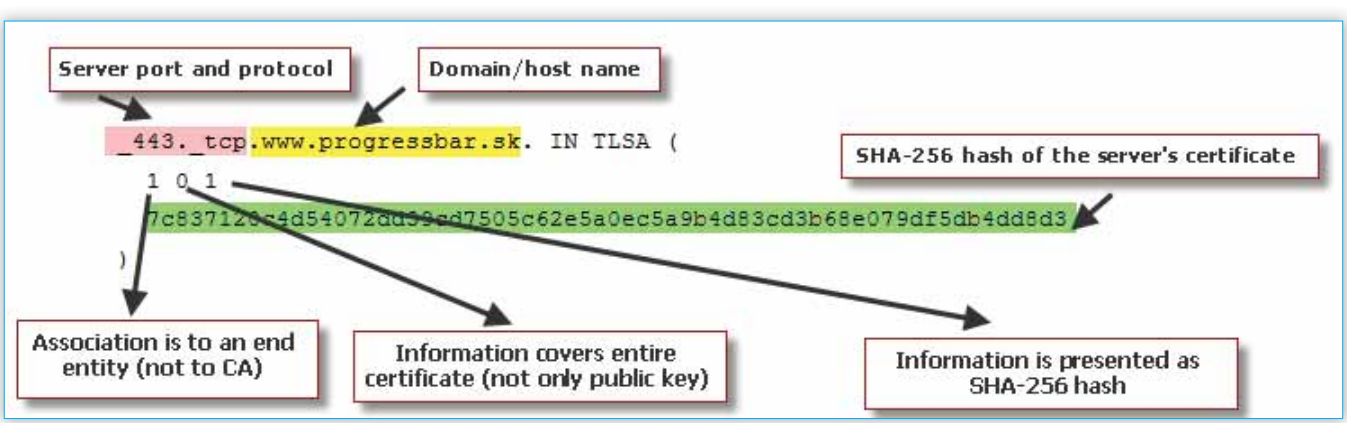
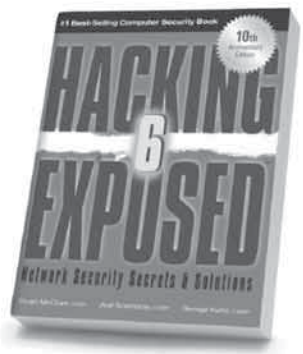


Figure 3. *Sample DANE TLSA RR*

Stop Hackers in Their Tracks



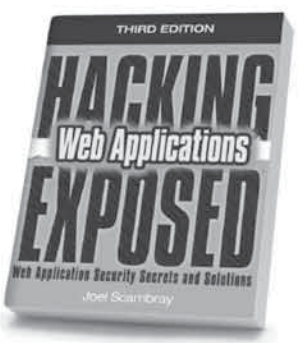
Hacking Exposed, 6th Edition

Hacking Exposed Malware & Rootkits

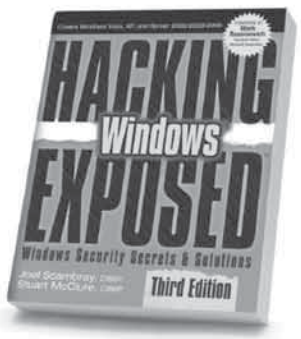
Hacking Exposed Computer Forensics, 2nd Edition

24 Deadly Sins of Software Security

Hacking Exposed Wireless, 2nd Edition

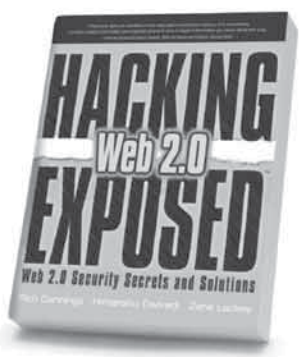


Hacking Exposed: Web Applications, 3rd Edition



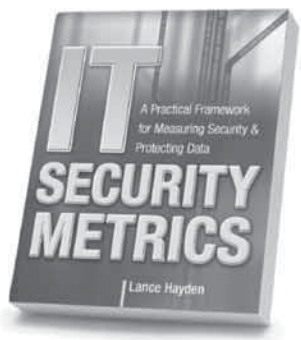
Hacking Exposed Windows, 3rd Edition

Hacking Exposed Linux, 3rd Edition

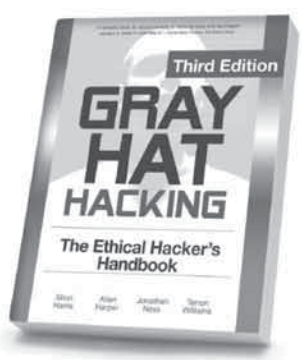


Hacking Exposed Web 2.0

IT Auditing, 2nd Edition



IT Security Metrics



Gray Hat Hacking, 3rd Edition

Available in print and ebook formats

Follow us on Twitter @MHComputing

Learn more. Do more.

MHPROFESSIONAL.COM

THE HASH FUNCTION CRISIS AND ITS SOLUTION

BART PRENEEL

Since the early 1990s, hash functions are the workhorses of modern cryptography. They are used in hundreds of applications that include password protection, code signing and digital cash. Many of the most widely used hash functions have been badly broken, which means that they do not deliver the security properties claimed. These attacks are not theoretical, but they allow to undermine real applications such as the security of certificates issued by CAs (Certification Authorities). This article reviews the problems with our current hash functions and looks at the solutions.

Cryptographic hash functions map input strings of arbitrary length to short output strings (see Fig. 1). Unlike all the other cryptographic algorithms, no key or secret value is involved in their definition. Hash functions are used in a broad range of applications: to compute a short unique identifier of a string (e.g. for digitally signing a document or code in combination with a digital signature scheme), as one-way function to hide a string (e.g. for the protection of passwords or passphrases), to commit to a string in a cryptographic protocol, for key derivation (e.g., to compute an AES key from a key agreed with the Diffie-Hellman protocol) and for entropy extraction in pseudo-random bit generators. As very fast hash functions became available in the early 1990s, cryptographers started to design other primitives such as stream ciphers, block ciphers and MAC algorithms based on hash functions. The HMAC construction is perhaps the most successful example, as it is widely used in protocols such as IPsec, SSH, and SSL/TLS.

The first proposal to use hash functions in cryptography can be traced back to the 1976 seminal paper of Diffie and Hellman on public-key cryptography. Between 1976 and 1996, about 100 designs of hash functions have been proposed. Most of them have been broken, frequently even within a few months or even weeks after the publication of the design. Three of these hash functions, namely MD4, MD5 and the US government standard SHA-1 became very popular; as an example, in 2004 Microsoft Windows had 800 uses of the hash function MD5. Unfortunately, security analysis has demonstrated that the above three hash functions are insecure as well; MD4 and MD5 are particularly weak, since they can be broken in microseconds.

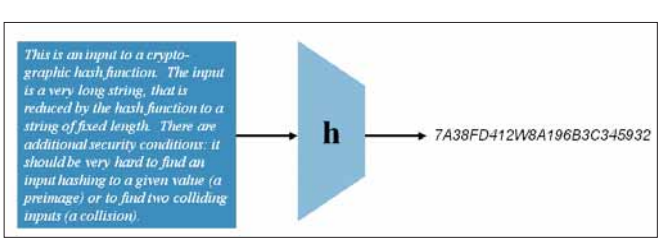


Figure 1. *A cryptographic hash function*

This article reviews the security requirements for hash functions. Next it explains why MD4, MD5 and SHA-1 are so widespread, discusses the weaknesses found in these functions and how these affect applications. We conclude by explaining what the solutions are to the hash function crisis: one can make some modifications to the applications or upgrade to SHA-2, or wait for the outcome of the SHA-3 competition.

Security Properties of Hash Functions

Cryptographic hash functions require three main security properties (see Fig. 2).

- **One-wayness or preimage resistance:** given a hash result y=h(x) it should be hard to find any input x' that maps to y. This property is required when one stores in a computer system the hash value of a secret password or passphrase rather than the value itself. The assumption is that an attacker may obtain the list of hash values (in UNIX system this list is stored in etc/passwd) but that this should not reveal the passwords.
- **Second preimage resistance:** given an input x and its hash result y=h(x) it should be hard to find a second distinct input x'

that maps to the same value y. This property is required when one has a digitally signed document of the form (x,Sig(h(x))), where Sig(.) is computed using a secret signing key (e.g. the secret key for the RSA signature algorithm). Assume that an attacker wants to modify the document to x' without knowing the secret signing key, and obtain the signed document (x',Sig(h(x')). If h(x')=h(x), the signature on x' will be the same as the signature on x and thus a signature on x' can be forged without knowing the secret key.

- **Collision resistance:** it should be hard to find two distinct inputs x and x' such that h(x)=h(x'). Assume that an attacker who writes device drivers wants to use them to spread malware. In order to prevent this, the operating system vendor checks the device drivers; if they are clean they will be digitally signed; every copy of the operating system checks the digital signature before installing a new device driver. The attacker can defeat this measure by creating two versions of the device driver, namely a clean version x and a version with malware x' with the property that h(x)=h(x'). He can now submit x for inspection, and obtain the signature Sig(h(x)). Later on he can use the same signature to distribute the version x' with the malware. Collision resistance is also needed if one party commits in a protocol to a secret value x by sending h(x||r) to the other party, where r is a random string.

At first sight, second preimage resistance and collision resistance seem very similar: the result is that an attacker has two distinct messages with the same hash value. However, finding collisions is much easier than finding second preimages, because an attacker has much more freedom in a collision attack: he can freely choose both messages, while for second preimages the first message is fixed. For a flawless hash function with an n-bit result, finding a preimage or a second preimage takes about 2n hash function evaluations, while finding a collision requires only 2n/2 hash function evaluations. The reason for this is known as the birthday paradox: for a group of 23 people, the probability that two people have the same birthday is about 50%. The explanation is that such a group has 23*22/2 = 253 distinct pairs of people. On the other hand, a group of 182 people is needed to have a probability of 50% to have someone with a birthday on any given date.

While one typically considers in cryptography individual problems, solving one out of multiple instances can be a lot easier. If one has 2t inputs, finding a second preimage for any of the values requires only 2n-t hash function evaluations; a similar observation holds for preimages. This problem can be solved by randomizing a hash function: every instance is made unique with a second randomly chosen input. In the context of UNIX passwords this randomizing parameter is called a `salt'.

In practice, one uses for (second) preimage resistance a hash function with at least n=128 bits. Even if one can attack 1 billion hash values in parallel (t=30), finding a (second) preimage within 1 year requires more than 10 trillion US$. On the other hand, one could find for such a hash function a collision in a few hours for 1 million US$. For long term collision resistance, a hash result of at least 256 bits is required.

In the past years other security properties have been identified, such as indistinguishability from a random oracle; however, the detailed discussion of these technical properties is beyond the scope of this article.

The rise and fall of MD4, MD5 and SHA-1

The first generation of hash functions was designed during the 1980s; many schemes were broken, and it was only near the end of the decade that the first theoretical results appeared. Around 1990, a very important developed occurred in cryptography: until then, most cryptographic algorithms were implemented in hardware, either in dedicated boxes to encrypt network communications or in hardware security modules to protect sensitive information on computers. As PCs became more powerful, and got connected to LANs and later on to the Internet, there was a growing need to implement cryptographic algorithms in software. However, the symmetric algorithms available at that time such as DES and LFSR-based stream ciphers were designed to be efficient and compact in hardware. In order to solve this problem, researchers started proposing new cryptographic algorithms that were more suitable to software implementations, such as the Snefru (from Merkle, who invented public key agreement in the mid 1970s) and MD4 and MD5 (from Rivest, the R in the RSA algorithm). Around the same time, Biham and Shamir invented differential cryptanalysis and managed to break DES (with a theoretical shortcut attack) and FEAL-8 (with a very efficient attack); Snefru, based on large tables, turned out to be vulnerable to this powerful technique, but MD4 and MD5 held up remarkably well. Both algorithms used addition mod 232, XOR, and bitwise operations, which were extremely efficient on the upcoming 32-bit RISC architectures. Overall, these algorithms were about 10 times faster than DES, which was a crucial advantage in the early 1990s. In addition, free source code for both algorithms was made available in 1991 and the algorithms and the code could be freely used (unlikely Snefru that was patented). The RFCs 1320 and 1321 containing the code were both published in 1992. At the time all algorithms and code for encryption and decryption was tightly controlled by export laws; the restrictions on export of hash functions were less strict. All these elements contributed to the enormous popularity of MD4 and MD5 and can help to explain why Microsoft Windows had 800 uses of MD5. Internet protocols such as APOP, IPsec, SSH, SSL/TLS all use MD5 (and sometimes MD4). For authenticating network packets, a hash function had to be turned into a MAC algorithm, that takes as second input a shared secret key K. After failures of attempts such as the secret prefix method, h(K||x), the secret suffix method h(x || K), and the secret envelope method h(K || x || K), the standardized solution was HMAC, defined as $MAC_K(x)$ = h(h(K ↰ ipad || x) ↰ opad), where ipad and opad are fixed strings. Note that APOP uses the secret suffix method based on MD5.

Very quickly after the publication of MD4 in 1990, it became apparent that with 48 simple steps its security margin was very small; this prompted Rivest to design MD5, that had 25% more steps (namely 64), where each step had some extra operations. In 1996, Dobbertin found collisions for MD4 in 220 operations which is much faster than the design goal of 264; his attack used sophisticated improvements of differential cryptanalysis. Eight years later, Wang et al. showed how to further extend differential cryptanalysis in order

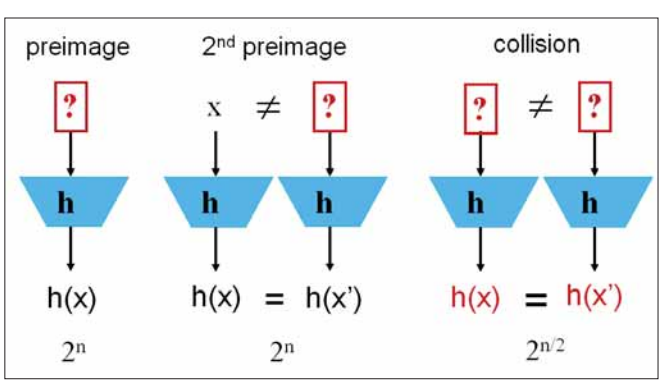


Figure 2. *Security properties of a cryptographic hash function*

to find collisions for MD4 in a few operations (by hand!). While MD5 was intended to be more secure, early results in 1993 and 1996 indicated that its security margin was very small; as a consequence a recommendation was issued in 1996 to stop using MD5 for applications that require collision resistance. In 2004, Wang et al. found collisions for MD5 in 15 minutes, again by using enhancements to differential attacks. Later on, these techniques were fine-tuned, resulting in collisions in microseconds. Stevens et al. managed to strengthen the techniques further; their work culminated in an attack in 2008 on a Certification Authority (CA) that still used MD5 to sign certificate; with the chosen prefix attack (also known as a correcting block attack), they managed to obtain a signature on a user public key that could also be used as a key for a rogue CA and thus impersonate any website on the internet. The attack was launched four years after the publication of the results by Wang et al., yet 6 CAs had still not upgraded their hash function. It is perhaps important to point out that the security of both MD4 and MD5 against brute force collision attacks (that do not require any knowledge of cryptanalysis) is 264 operations; this was already insufficient to protect against a motivated opponent in 2000. The best preimage attack for MD4 requires 2102 operations; this is less than the design goal of 2128, but still far beyond reach today; it has also been shown that for a small fraction of messages, finding second preimages is easy. The best known preimage attack on MD5 requires 2^{123} operations.

In 1993 NIST (the National Institute for Standards and Technology in the US) decided to standardize a hash function; they did not trust the security of MD4 and MD5 (perhaps based on their own cryptanalytic work) hence they proposed the Secure Hash Algorithm (SHA) designed by NSA. The SHA algorithm (today called SHA-0) had a 160-bit result, hence offering a security level of 280 operations against brute force collision attacks. It is more than twice slower than MD5. Two years later, SHA was withdrawn and a new version called SHA-1 was published; the reason was an attack identified by NSA that was never published. Later on, the academic community has discovered serious weaknesses in SHA-0; the best known attack today finds collisions in about 1 hour. In 2004, Wang et al. surprised the cryptographic community by showing a collision attack on SHA-1 that requires 2^{69} operations rather than 2^{80}. Several teams have since then announced improvements, but so far no one has managed to produce a collision for SHA-1 or a convincing description of an attack with complexity less than 2^{69} operations. The best result is a collision for SHA-1 reduced to 75 out of 80 steps that was found in November 2011. For second preimages, a theoretical attack shows that for up to 61 steps SHA-1 does not have perfect behavior.

Solutions to the hash function crisis

A first solution is to replace MD4, MD5 and SHA-1 by hash functions with a larger security margin that are currently standardized. If that is not possible, one has to carefully examine the application in which the hash function is used to evaluate whether the security is still adequate. A third solution is to wait for the new standard SHA-3 that will be selected in late 2012.

In 2002, NIST published the SHA-2 family of hash functions that intend to offer much higher security levels than SHA-1; the SHA-2 family has output results varying from 192 bits to 512 bits. For outputs of 192, 224, and 256 bits, the operations are on 32-bit words (as for SHA-1) and the number of steps is 64. For the larger output lengths (384 and 512), 64-bit words are used and the number of steps is increased to 80. The steps themselves have become more complex, which clearly enhances the security. On the other hand, SHA-2 is still based on a combination of additions, XORs and Boolean operations and the main non-linear component consists of carries, just as for the other members of the MD4 family. No document has been published that justifies the design decisions; as NSA has made some mistakes earlier with SHA-0 and SHA-1, this has cast some doubts on the design. After one decade, the conclusion is that SHA-2 has withstood the current attack techniques: the most powerful attack is an attack that demonstrates deviations from randomness for 47 out of 64 steps of SHA-256; collisions faster than the birthday paradox seem to be possible for 53 steps with current techniques. On 32-bit architectures SHA-2 is more than four times slower than MD5, but for 64-bit architectures this factor is reduced to two.

There are other alternatives to SHA-1 that have been standardized in ISO 10118-3 (but not by NIST): RIPEMD-160 is a hash function from 1996 with a 160-bit result; it is 20% slower than SHA-1 but seems to have a substantial security margin. Whirlpool offers a 512-bit results; it security margin is not as large as hoped for, but it is still an interesting alternative based on very different design principles.

If it is not possible to replace the hash function, one can examine whether or not collision resistance is needed. While hash functions are widely used, there are only two important applications where collision resistance is needed: digital signatures in which an attacker can freely choose both documents that are signed and protocols using commitments. The main commercial applications are code signing and digital certificates. NIST has published the RMX mode, in which the data to be signed is randomized by the signer, hence collision attacks are rendered useless. This mode may not be sufficient for MD4 and MD5 but SHA-1 is likely to possess the security properties to make this solution work. One caveat is that of course the signer himself can still defeat this mode by choosing the randomness prior to the message. Stevens has also published an ad hoc solution: the collisions found with the current attacks have a particular structure, and one could scan for messages with this structure and reject them. This method can likely be defeated by a clever opponent who creates a variant of the current collision attacks.

If the opponent does not have any control over the message to be signed (or the message has been signed before 2004), an opponent needs to launch a second preimage attack. While one can imagine that such an attack becomes feasible for MD4 in the next few years, for MD5 this is still beyond reach, and for SHA-1 there is still a substantial security margin.

On the Internet, the most popular application of MD4, MD5 and SHA-1 is the HMAC construction. For HMAC-MD4, the best known attack has complexity 2^{72} (in both texts and computation). HMAC-MD5 can only be broken in a related key setting, in which an opponent can compute MAC values for different keys that are unknown but related in a specific way; the complexity of this attack is 2^{51} texts and 2^{100} operations; if proper key management is used, related key attacks should not be a concern. In a regular attack setting only 33 out of 64 steps can be broken. For HMAC-SHA-1 only 53 out of 80 steps have been broken so far. The conclusion is that HMAC-MD4 should not longer be used; HMAC-MD5 should be phased out as soon as convenient, while HMAC-SHA-1 seems still acceptable for the next 5-10 years. For the secret suffix method in APOP, the situation is much worse: for MD4 and MD5 secret keys can be recovered with a few thousand chosen texts and with a few seconds of computation. The security of SHA-1 with APOP is likely to be insufficient as well.

In the last decade some new structural or generic attacks have been identified, that all apply to most hash functions designed before 2000, that are iterated hash functions with an internal state size equal to the output size. One of these attacks (by Joux) shows that if the result of two iterated hash functions are concatenated (that is $h(x) = h_1(x) \,||\, h_2(x)$) in order to get a much strong hash function,

the resulting function is only as secure as the strongest of the two components; in other words, the weaker hash function does not help but costs extra. As a consequence of these attacks, consensus grew around 2005 that there is a need for new hash functions that offer an adequate security margin for the next 30 years or more, and that it is unclear that any of the existing hash functions satisfy these requirements. This has motivated NIST to organize an open competition; this procedure has been used with great success in the past in symmetric cryptography (e.g. for the selection of the block cipher standard AES).

The NIST SHA-3 Competition

An open call was published on November 2, 2007 for a hash function SHA-3 that would be compatible in terms of parameters with SHA-2 (results from 192 to 512 bits). The winner of the competition needs to be available worldwide without royalties or other intellectual property restrictions. Preparing a submission required a substantial effort, yet NIST received 64 submissions. Early December 2008, NIST has announced that 51 designs have been selected for the first round. On July 24, 2009, NIST announced that 14 algorithms have been selected for the second round. On December 10, 2010, the five finalists were announced: Blake, Grøstl, JH, Keccak and Skein. Blake and Skein have a smaller internal state (although Skein has also a variant with a larger internal state) and both use the same operations as in MD4/MD5/SHA-1/SHA-2; moreover, the main building block is a kind of block cipher, while the other designs are built based on one (or two) permutations. Grøstl and JH have a medium size internal state and Keccak has a large one (200 bytes). Grøstl uses 8-bit S-boxes like AES, while JH and Keccak rely on smaller S-boxes (with 4 respectively 5 bits). In terms of performance, Blake and Skein seem to be more performant on high end processors, while Keccak is performing best in hardware; for embedded machines, all designs are slower than SHA-2. Keccak is the most original design, as it uses a new kind of construction called a sponge. For security, there is no clear picture yet. What is important to note is that all designs have been tweaked since their submission (in many cases rounds have been added to increase the security margin in response to attacks); some designs have been even changed twice.

A first observation is that the half-life of a hash function is about 9 months: by June 2008 half of the submissions were already broken. After this date, only strong functions remained (that were further improved), and the number of attacks has decreased. Most of the cryptanalysis work has been performed by European researchers; 3 of the 5 finalists have been designed in Europe, while the original 64 submissions had a much broader geographic spread. It is also interesting to point out that only 2 of the 64 submissions were based on a primitive the security of which could be reduced to a mathematical problem; as they were too slow, they were not selected for the second round. On the other hand, a large number of security reductions have been proven under the assumption that the underlying building block (such as a block cipher or a permutation) is ideal.

Security and performance updates on the SHA-3 competition can be found in the SHA-3 Zoo and eBASH websites that are maintained by the ECRYPT II project (*http://www.ecrypt.eu.org*).

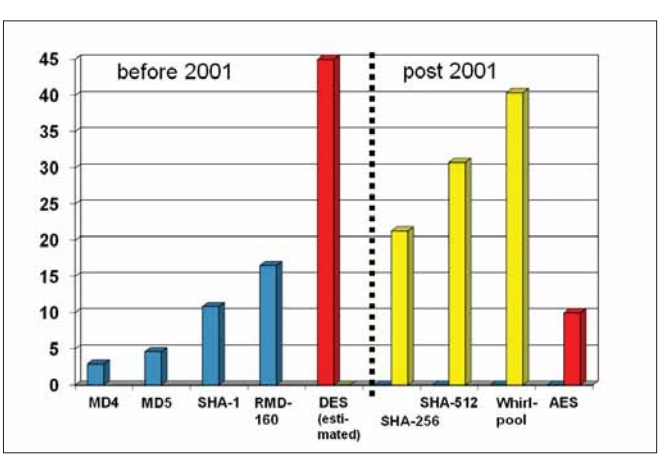


Figure 2. *Performance in cycles/byte of the hash functions MD4, MD5, SHA-1, RIPEMD-160, SHA-256, SHA-512, Whirlpool and the block ciphers DES and AES on an AMD Intel Pentium D 2992 MHz (f64) [source: http://bench.cr.yp.to/index.html]*

Conclusions

We have witnessed a cryptographic meltdown in terms of collision resistance of widely used hash functions: schemes that were believed to be secure could be broken in milliseconds. Fortunately the implications of this meltdown have been very limited, because very few applications rely on collision resistance. For second preimage resistance and for constructions such as HMAC, the attacks have been less dramatic, but replacing MD4 and MD5 is essential.

One can be confident that the new SHA-3 algorithm will have a solid security margin and a good performance, even if it may be slower in some environments than SHA-2. Even if the SHA-3 design reflects the state of the art in 2008, there have been substantial advances in the theory of hash functions and our understanding today is much better than 10 years ago. Developers should start to plan an upgrade to SHA-3 by the end of 2012 or in early 2013.

Finally, application developers need to rethink how they use cryptography. In the early 1990s, the hash functions MD4 and MD5 were more than 10 times faster than DES and they were (wrongly) believed to be also much more secure. This explains why most cryptographic applications (both for network and computer security) prefer hash functions over block ciphers. An example of this is the use of HMAC rather than CBC-MAC. Today the roles are reversed: block ciphers are faster than hash functions, hence if performance is a concern block ciphers should be preferred. On modern processors, AES in software is six times faster than DES, while SHA-3 is likely to be two to three times slower than MD5, hence block ciphers are about twice faster than hash functions (on 64-bit machines the factor may be a bit smaller). This is illustrated in Fig. 3, that presents the performance of hash functions and block ciphers on AMD Intel Pentium D. Moreover, since 2010 high end Intel processors have dedicated AES instructions that give a speedup of a factor up to 10. This will further increase the advantage of AES, at least until special instructions are added for SHA-3.

While one can expect SHA-3 to be used for the next two decades, cryptographers will still keep looking for new hash function designs: one challenge is to design lightweight hash functions for environments with limited resources (power, energy, area); another problem is the design of hash functions with solid security proofs.

AUTHOR'S BIO

Prof. Bart Preneel received the Electr. Eng. and Ph.D. degrees from the University of Leuven (Belgium) in 1987 and 1993. He is a full professor in the COSIC research group at the University of Leuven. He has authored more than 400 scientific publications and is inventor of 3 patents. His main research interests are cryptography and information security and he frequently consults on these topics. He is president of the IACR (International Association for Cryptologic Research). He has served as program chair of 14 international conferences and he has been invited speaker at more than 70 conferences in 30 countries. In 2003, he has received the European Information Security Award in the area of academic research, and he received an honorary Certified Information Security Manager (CISM) designation by the Information Systems Audit and Control Association (ISACA).

WEB APP CRYPTOLOGY

TRAVIS H.

Those who cannot learn from history are doomed to repeat it.

George Santayana

Most data will be encrypted, and unencrypted data will be the exception very soon. Cryptography and safety engineering are different, but they share the same principle; that we do not know how to design efficient, practical, secure systems, but we know how some have failed in the past. As encryption becomes ubiquitous and more people rely on it, we shall find ourselves in a similar situation to elevator, fire safety, and aviation engineers; that human lives will depends on us doing our job properly. However, unlike these physical analogies, we will face evolving threats, and virtually nobody will understand our job, if and how we failed, or the consequences of our failure. Thus we must learn from the failures of the past, or be doomed to repeat them.

The problem with bad security is that it looks just like good security. You can't tell the difference by looking at the finished product. Both make the same security claims; both have the same functionality […] Yet one is secure and the other is insecure.

– Bruce Schneier

This article will be a review of how web applications have used cryptography improperly in the past, which led to the compromise of a security property. By reading this, web application developers should learn certain mistakes to avoid, penetration testers may learn common mistake patterns for which they can test, and those not familiar with cryptology may gain a new appreciation for the subtlety and attention to detail required.

Body

Why Web Apps Need Crypto

If we ignore TLS (which is handled by the web server), cryptography is usually needed in web applications due to the stateless nature of most RESTful web applications. That is, the session state is held in the client, in the form of a URL (including GET parameters), POST parameters, and cookies (especially login cookies2, sometimes called authentication cookies). It is possible to store client state on the server side, but it introduces complications in:

- web app programming
- performance (disk I/O bottleneck)
- load-balancing across multiple web servers
- geographic redundancy and resiliency to network failures

Login Cookies

We most often need a secure session token as the foundation of web app security, even if we are storing state on the server side, since we need to look up that state somehow. There have to be enough states to be difficult to enumerate by brute force, and it must lack a guessable pattern; cryptographers have a measure of this, called guessing entropy:

$$G(X) = \sum_{x=0}^{\max(R(X))} P(X=x)\,(x+1).$$

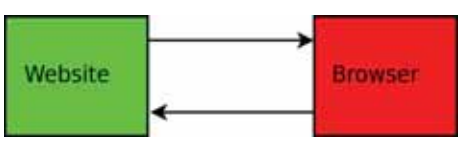
This state is usually stored as a base64-encoded value in a HTTP cookie. It is used to indicate that the request is being sent from a browser which is logged in. Of course this cookie is sent with every request to the site, so it may come from an image link on another tab, and so login cookies are not protection against CSRF attacks. Generally speaking, it includes or implies identity (login name), and is usually used instead of HTTP authentication for various reasons (see the IETF WG for HTTP auth if you want to help fix that).

Application of Cryptography

Since we will (on occasion) send data to the client which we wish to have returned unaltered, and we wish it not to be forged outright, we need cryptography. We may or may not need it to also remain confidential. This is analogous to sending data across a wire which is controlled by the adversary, and not wanting it to be altered or spoofed, and possibly wanting it to remain private as well.

Most communication encryption does something like this:

But in our case, we are protecting data we want sent back to ourselves – you can think of it as sending a message to your future self, as relayed through the browser:



Since we do not know how to send data back through time yet, whatever system we use must be a unidirectional protocol, which is also required for email encryption, since email is nominally a *store and forward* network. This implies that they are vulnerable to replay attacks, which can be mitigated in a number of ways. The easiest countermeasure in this case is to use a timestamp (assuming that it is monotonically increasing) to ensure *freshness*. Normally this requires storing the peer's last timestamp, or synchronized clocks, but we are assured synchronization since we are communicating to our (future) selves.

In this manner, we are using the web client (browser) as a storage device. This is convenient since if it fails (i.e. the browser state is cleared), the user won't expect it to work. Since it is outside our control and trust boundary, it should be protected with cryptography. But since this is so subtle, many organizations get it wrong.

The WSJ.com Mistakes

Unix crypt(3)

First, let me give you a little background on the Unix crypt1 function. That number in parenthesis shows that it's a library function, since it was described in section 3 of the original Unix manuals. While it may have started as a bidirectional encryption routine, it ended up as a one-way function2 for some very good reasons. At the time, it was really close to DES3 encryption with a plaintext of all zeroes, using the input as the key, which is swapped from how one normally uses ciphers, but seemed sufficient as a one-way function at the time, since it seemed difficult to determine the key given the ciphertext.

Cooking with Salt

When designing the software crypt(3) routine, cryptographers did not want people to be able to easily use off-the-shelf hardware to brute-force the algorithm, so they used 12 bits of Salt to change the algorithm to be incompatible.

How crypt(3) Worked

1. User's password truncated to 8 characters, and those are coerced down to only 7 bits ea.
2. Forms 56-bit DES key
3. Salt used to make part of the encryption routine different
4. That is then used to encrypt an all-bits-zero block: crypt(s,x)=s+E_{\text{x}}(\overline{0})
5. Iterate 24 more times, each time encrypting the results from the last round; this makes it slower (on purpose)

WSJ.com Attack #1

At first they simply used crypt(3) on the username concatenated with a secret string, and used that as the authenticator:

- Unix crypt (salt, username + secret string) = salt + encrypted_data = WSJ.com login cookie
- The problem here is that crypt(3) only hashes 8 octets (characters), and so it truncated the input string. crypt(s,"dandylionSECRET") = crypt(s,"dandylio")

Thus, an adversary could pick an 8 character (or more) username, a salt s, do crypt themselves, and have a valid login cookie without ever needing to know the secret.

WSJ.com Attack #2

The next problem was that usernames identical in the first 8 letters had identical login cookies, so adversaries could tell the salt was the same. Thus, if two usernames began with the same 8 characters, the adversary could use the same authenticator for the other user's account (presumably the username was a different value in the cookie).

WSJ.com Attack #3

The next attack is tricky, but very powerful. It is a form of an *adaptive chosen-message attack*, and is used to attack each octet of the secret individually.

In this attack, the adversary registers a seven-character username, and guesses at the first octet of the secret (the last character of the eight that crypt honors). One of the 128 possibilities (one bit is lost) will yield a valid authenticator for this user (in this case, since the salt was fixed, he could test it against one he got by logging in normally).

Then, and this is the key, he could attack the second byte of the octet by registering a six-character username. The attack proceeds like an inductive proof.

The take-away from this is that if each attempt takes one second, and the adversary had to guess every octet individually, it would take 2x10^9 years, but if he can attack each octet separately, he can do it in 17 minutes. The adaptive attack makes the solution linear time instead of exponential.

The secret was "March20".

Common Cryptographic Mistakes

Poor Random Number Generation

All cryptography relies upon generating numbers in a way that the adversary cannot predict. In September of 1995, Ian Gold-

berg posted a tool for predicting the PRNG in Netscape's SSL implementation to the cypherpunks mailing list. In November 2007, Leo Dorrendorf et. al. from Hebrew University of Jerusalem and University of Haifa published a weakness in Microsoft's CryptGenRandom. In August of 2007, Dan Shumow and Niels Ferguson of Microsoft published a weakness in Dual EC DBRG. In May, 2008, security researcher Luciano Bello found a major flaw in the way that Debian's OpenSSL patch affected the PRNG. In December of 2010, a group calling itself failOverflow announced recovery of the ECDSA key used by a major game company for signing software for their game console.

Thus, random number generation is the *sina qua non* of cryptography. However, it is still a black art, and relatively little attention is paid to RNG design. We have no way of proving that random numbers are not predictable (that they lack a pattern) through testing the output, because every output stream is just as likely as any other. Even if we could, since there are an infinite number of possible patterns, so we cannot test for all of them.

Hashes

Cryptographic hashes are one-way functions; given the input, it's easy to calculate the output, but not vice-versa. This is known as *preimage resistance*. There are actually other properties of an ideal hash, but I cannot go into them in this limited space. They are often used to protect passphrases so that obtaining the passphrase database does not compromise the user's passphrase.

A common mistake when using hashes is to allow users to pick from a small, relatively predictable input space, and then hash that without any salt or *uniquification vector*, which is a term I made up to describe the idea of making every entry unique. In this case, it is relatively easy to test a dictionary of probable passwords against the list. In many cases, you can download *rainbow tables* of likely passwords for these hashes. Rainbow tables are just clever ways of storing tables of precomputed hashes. Similarly, you can also google each hash and look for people posting what its preimage is.

I recommend that if you use a *n*-bit hash, that you use a n-bit uniquification vector to saturate the hash's input space. But better than that, you have HMAC (which involves a secret only the server knows), PBKDF2, and scrypt for passphrase protection.

ECB Mode

Perhaps the most common mistake in cryptography is to use Electronic Code Book (ECB) mode. If you encrypt each block of plaintext independently of all the others, you leave yourself vulnerable to several attacks.

First, the adversary can determine the block size of the encryption, and then swap blocks as he likes, without ever knowing the key. A lay person can understand the consequences of this if he imagines an adversary being able to swap parts of a bank transaction, such as the sender and recipient account numbers. This is a major problem with *integrity*.

The second major problem is one of confidentiality; even if I don't know the key, I can see that certain blocks repeat. Thus, macroscopic patterns above the block level may be visible. This is hard to explain in words, so allow me a "graphic" example.

Let us assume this is our plaintext:

Then this is the ECB encryption of that image:

It is obvious even to a non-cryptanalyst that this is a picture of Tux. For comparison, most of the other block cipher modes would make it look like this:

CBC Mode Mistakes

Chained block cipher mode is the most common block cipher mode. In this mode, the output of the block cipher function is exclusive-or'ed (XOR) with the next block's plaintext prior to encryption. In other words, it takes the previous ciphertext block, XORs it with the plaintext, and encrypts that. In order to jumpstart this process, there's an *initialization vector* (IV) which is used on the first block, since there is no previous ciphertext block.

The whole point of the chaining is that it makes each encryption of the same data different. But many sites make the mistake of using the same IV. Generally the key is the same, and so if the plaintext is the same, it encrypts to the same ciphertext. If this was encrypting, say, a password database, it would reveal that two passwords were the same.

Using Encryption Instead of Integrity Protection

When most people think of cryptography, they think of encryption. Encryption protects the confidentiality of the plaintext, but your login cookies probably don't need to be confidential. They do need to be unforgeable, and you need to be able to detect if they've been modified (both properties are related). With certain exceptions, encryption is almost always the wrong tool for this, as fiddling with the ciphertext will corrupt at least one block, which may be decrypted and syntax-checked too late to matter. If you're lucky, that corrupted block will not be a syntactically-valid string, and your software will reject the entire message on that basis.

Message Authentication Codes

Shallow men believe in luck. Strong men believe in cause and effect.

Ralph Waldo Emerson

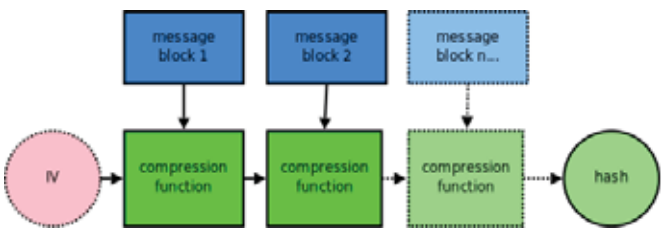
We want a way to verify that the data have been tampered with. A hash of the data, attached to the data, isn't enough, since the adversary could modify the data and add a new hash. What we need is something like a hash, but that requires a secret (key) to compute.

CBC-MAC

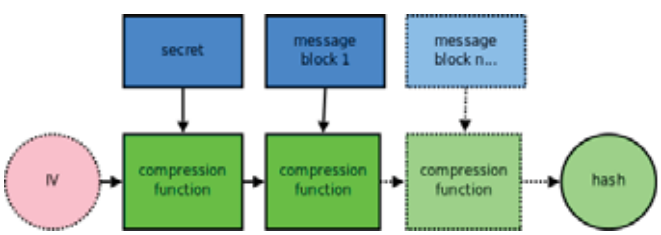
In CBC-MAC you encrypt the message in CBC (or CFB) mode, and then encrypt the last block once (for good measure), and use that as the "keyed hash". This is specified in a number of government standards, but has problems with malleability.

One-Way Hash Function MACs

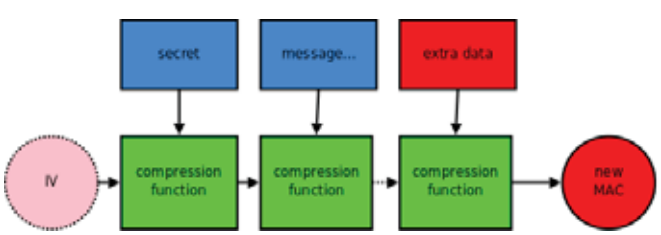
Suppose you wish to simply concatenate a key and the message, then hash it using a standard cryptographic hash. This is the obvious solution to create a MAC but has several subtle problems. First, let's examine how hash functions work. Many are "iterative", so that they can work on any sized input, yet only carry a small amount of state from one iteration to the next.



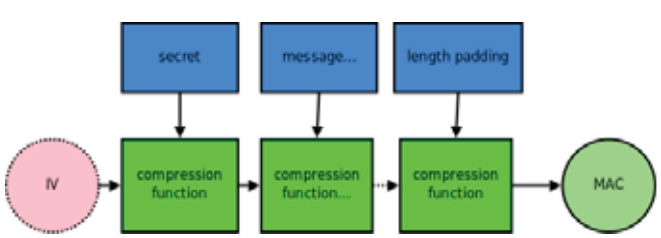
In most constructions, the IV is fixed in the hash specification. Now suppose we're going to use a secret which just happens to be one block size (without loss of generality).



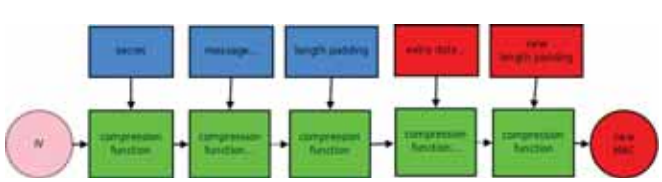
The problem is that anyone with the MAC for a message can tack data on to the message and generate a new MAC:



One way that hashes are strengthened against this kind of thing is to use Merkle-Damgaard Strengthening, which adds the length to the end of the message (since one may be calculating it on the fly from a variable-length stream, presumably):



The problem with this is that an adversary can tack more data and a new length onto it:



When I brought the web application community's attention to this in 2009, it was just theoretical; I wasn't aware of any vulnerabilities, but in September of 2009, Netifera found this vulnerability in the Flickr API.

One problem with everything up until this point is that the output of the last iteration of the compression function is the MAC, which leads to hash extension attacks. If we had a one-way function to "finalize" the hash, it'd be much more difficult to "work backwards" through it to continue iterations of the hash. That is why Bruce Schneier (in Practical Cryptography) suggests doing hashes twice before using them anywhere. That is, if you're doing SHA512, then do it another time on the output, and the second round prevents anyone from adding more data to the initial hash.

There are other questionable one-way hash function MACs, such as prepending the message length and secret, or putting the secret at the end of the hash (which is vulnerable to malleability if a collision is found), or putting secret(s) at beginning and end. Still others involve two hashes, padding, or putting part of the key into every message block, but nothing we've covered so far is really worth considering since we have something with a proof of security.

HMAC

In HMAC, we take a key K, XOR it with 0x363636...3636, concatenate the message to that, hash it, then prepend (K XOR 0x5c5c5c...5c5c), and hash it all again. This was defined and analyzed by Bellare, Canetti, and Krawczyk in 1996, and formalized as RFC 2104 the next year. It's not entirely clear without reading their paper why it is so strong, but it is. Since it does not seem to be affected much by collisions in the underlying hash, HMAC-MD5 appears to still be secure (though I would not recommend it for a new design). It is very interesting that a construction would rely on fewer properties of the underlying components than they would on their own; this is like building a house out of walls that could not stand by themselves.

Another use for HMAC is to derive multiple keys from one master key k. In this, you could compute e.g. HMAC(*k*, "1") for one key, and HMAC(*k*, "2") for another key. The messages encrypted are unimportant, as long as they are distinct. These two keys, perhaps one for encryption and one for authentication, would be computationally secure from each other and from the original key. That is, if a weakness in the encryption led to the recovery of the encryption key, it could not (under standard assumptions) compromise the master key from which it was generated, nor would it compromise the authentication key.

No Public Key Needed

It is futile to do with more things that which can be done with fewer.

– William of Ockham (*c.* 1285–1349),
Summa Totius Logicae

While a digital signature sounds like what we want, it is an asymmetric algorithm, and so is unnecessarily slow, requires (comparatively) large keys, and still requires some sort of hash algorithm anyway. What we really need is HMAC, which is the symmetric (one key) version of a digital signature.

This is obvious once we realize that we are just sending data to our (future) selves; once we realize that, it follows that the advantages of public key's key management are irrelevant, and using a public key algorithm would only introduce complexity and potential weakness into the system. Public key is done by

the web server, in SSL/TLS, but that is not normally your concern as a web application developer.

Security Property	Symmetric Algorithms	Asymmetric Algorithms
Confidentiality	Cipher	Public key encryption
Integrity	MAC	Digital signature

Wordpress Cookie Integrity Protection Vulnerability
Wordpress had a very interesting vulnerability in its login cookies.

- let | be a seperator character of some kind, and let + be concatenation
- let MAC = HMAC-MD5_K(USERNAME + EXPIRY_TIME)
- authenticator = USERNAME + | + EXPIRY_TIME + | + MAC

In this case, the expiration time was in seconds since the epoch, and you should know that Wordpress is written in a scripting language (PHP). The vulnerability is quite subtle; the MAC did not have a seperator between the username and expiration time. So one could register a valid user named "admin0", get a login cookie for that user, and then calculate a new authenticator for admin:

- authenticator = "admin" + | + time_1 + | + HMAC-MD5_K("admin0"+time_2)

This is possible because of ambiguous formatting and the way PHP handles leading zeroes when parsing a numeric value.

While adding a separator may solve this particular problem, the core solution of unambiguous parsings is important enough that many certificates and cryptographic routines use some kind of encoding rule set for Abstract Syntax Notation One.

Conclusion

Do not meddle with the products of cryptographers, for they are subtle and hard to understand.

Cryptography is quite possibly the most nuanced and difficult specialty in security. Computer security is quite possibly the most complex topic in computer science today. But here are some simple heuristics:

- Don't use ECB mode
- Don't use stream ciphers such as RC4
- Don't use MD5, or even SHA-1
- Don't reuse keys for different purposes
- Don't use fixed salts or IVs
- Don't roll your own cipher
- Don't rely on secrecy of a system
- Don't use guessable values as random numbers or PRNG seeds
- Use AES256 mode for encryption (CBC mode unless you're mixing data sources)
- Use HMAC-SHA512 for integrity protection
- Use SHA-512 with salt for hashing
- Use PBKDF2 for stored passwords or key derivation
- Use /dev/urandom on Unix
- Use RtlGenRandom/CryptGenRandom from ADVAPI32.DLL on MSWin

TRAVIS H

Travis H is the founder of the Bay Area Hacker's Association, and has been employed doing security or cryptography for financial institutions, top 50 web sites, e-commerce hosting companies, web software companies, and other organizations. He has been part of the largest security monitoring operation in the world, part of the security team for the most widely used piece of software in the world, and helped design an intrusion detection system. He has written a book on security2 which is available free online. He's also a bit uncomfortable tooting his own horn for he is keenly aware of his own ignorance, and you can tell this because it's the only place he uses the third person.

He is currently evaluating new employment opportunities, so you could tell him about a job you have, if it's really challenging and interesting.

CODENAME:
SAMURAI SKILLS COURSE

<< Penetration Test Training Samurai Skills >>

- You will learn Real World Hacking Techniques for Targeting , Attacking , Penetrating your target
- Real Live Targets (Websites , Networks , Servers) and some vmware images
- Course Instructors are Real Ethical Hackers With more than 7
- years Experience in Penetration Testing
- ONE Year Support in Forums and Tickets
- Every Month New Videos (Course Updated Regularly)
- Suitable Course Price for ONE Year Support
- Take Our course at your own pace (any time , any where)
- Our Course is Totally Different from Other Courses (new Techniques)

QUANTUM KEY DISTRIBUTION FOR NEXT GENERATION NETWORKS

SOLANGE GHERNAOUTI-HÉLIE, THOMAS LÄNGER

The generation and distribution of cryptographic keys constitute a major weakness of all currently commercially available cryptographic solutions. Accordingly, when organizations deal with critical and confidential data, truly reliable mechanisms for the transmission of secrets need to be employed. This article demonstrates how Quantum Key Distribution (QKD) will contribute towards answering this need and reinforcing the confidence of security managers in cryptographic mechanisms.

The aim of this paper is not to describe how QKD works but rather to demonstrate the importance for ICT, risk, and security managers in depending upon more reliable cryptographic solutions that often are not particularly well understood from a technical perspective. The significance of the focus on the usability of QKD is underlined by succinctly presenting some real-life examples of use and by emphasising the potential added value when QKD is integrated into network architectures in native mode.

Information Security Management in modern organisations

Managing information security in a dynamic and occasionally chaotic operational environment can be a very difficult task for security practitioners. To reduce the complexity of the management task, managers have to depend upon reliable technical tools. Quantum key distribution (QKD) can provide a partial answer, particularly with respect to the confidentiality constraint (figure 1).

A risk-management process has to include consideration of all risk components in order to choose or to propose the most appropriate countermeasures. As a consequence of being concerned by the costs that countermeasures generate, security managers have to perform cost-benefit analyses in order to spend their limited resources in the most appropriate way to generate the best possible results. This requires proactivity and means reducing risks and their impacts by decreasing the number of vulnerabilities. To mention only two examples of vulnerabilities, nowadays security managers need to be aware that SHA-1 was broken in 2005, and that the RSA SecureID two-factor authentication solution was compromised in 2011. Reducing cryptographic vulnerabilities is therefore becoming a crucial part of any risk management and information security approach. In the same way, security managers should consider some of the following questions when planning security investment:

- Will confidential data encrypted by classical cryptographic mechanisms still remain confidential in the long term?
- How long will confidential data that are considered as secure today actually stay confidential?
- How long could encrypted data stay secure, given that secret keys can be compromised?
- How can genuinely random numbers be generated to create secret keys that by definition will have a higher level of non-predictability key and thus help avoid replay attacks?
- How can threats against cryptographic mechanisms be diminished?
- How can vulnerabilities related to data confidentiality be decreased?

The answers of these questions can be found by integrating quantum technologies into existing cryptographic mechanisms, in order to minimize the problem of the creation and distribution of keys which, until today, has been the most crucial problem that the cryptography community has had to face when proposing secure tools for ensuring confidentiality over unreliable transmission systems.

Quantum Key Distribution (QKD) as a facilitator for information security managers

Currently, unauthorised third parties routinely and systematically attack communications and data transmitted over public networks. Even encrypted data can have their confidentiality breached and are completely vulnerable to the cryptanalytic power and determination of some specific actors. Every day, attempts are made to degrade strong cryptography into weak. At the same time, confidence in public key infrastructure or certification authorities (CA) can only ever be relative (as digital certificates can be corrupted or forged, or, even worse, illegally issued by a dishonest CA, key escrow recovery systems exist). Even cryptographic procedures that are currently considered to be secure are becoming increasingly vulnerable with the development of computing power and capacity and cryptographic know-how.

Public key cryptography, which is widely used at the moment and the security of which is based on algorithmic and mathematical complexity, could be under threat. It could become obsolete if advantage is taken of the virtualisation of cryptographic processes through cloud facilities exploited by malevolent entities. In light of this, it is incumbent upon technique- and risk-aware organisations to constantly be looking for up-to-date methods to protect their digital assets in order to be competitive and to ensure both business continuity and their reputations. Enterprises that use cryptography to protect against well-funded threats will need to develop mitigation plans in respect of stronger techniques, on the basis of appropriate cost-benefit analyses.

An issue that has to be considered in the area of ICT security management is that security managers have to deal with multiple subjects (technical, economical, ethical, legal, and managerial), which often means that multiple issues are resolved in multiple ways. Security issues are some of the deepest organizational issues, requiring the mobilization of elements of all the organization's resources and the involvement of other parties.

ICT security management has to work on (figure 1):

- Many stakeholders (policy team, compliance department, human resources department, IT department, etc.) all of whom are concerned with business functions and procedures;
- Some operational stages which include the implementation and testing of controls, physical and organizational safeguards, incident handling involving developers, the system administrators response team, project teams, etc.
- A number of auditing, evaluation, knowledge, and awareness processes in which auditors, trainers, experts, etc are involved.

In common with other pervasive domains of internal control, the creation and maintenance of the security architecture has to integrate a great number of components such as technical, human, organizational, and legal elements. It has to fulfil a number of functions located in many levels and using a wide range of the organisation's skills and resources.

Moreover, a number of regulations have emerged in several sectors of activity that oblige conformity from institutions. The requirement to respect various legal constraints in respect of information technology security reinforces the need to implement technical security measures that contribute to minimizing the legal risk taken by the institution when dealing with digital information.

Without pretending to be exhaustive, mention can be made for example of the Sarbanes-Oxley Act, the Basel II Agreement, the Gramm-Leach-Bliley Act (GLBA), the Health Insurance Portability and Accountability Act (HIPAA) or the EU's Privacy and Electronic Communications (Directive 2002/58) and Data Protection Directive (Directive 95/46/EC), EuroSox, etc.

The accountability, confidentiality, and integrity of data are required in several regulations, especially for financial institutions, although without specifying the kind of technology to be used to satisfy these criteria or the kind of security technology fulfilling these requirements. Most often, regulations refer only to *protecting informational assets in the best way,* a requirement open to different interpretations.

Many kinds of organizations are concerned by these regulations that define various requirements in respect of information

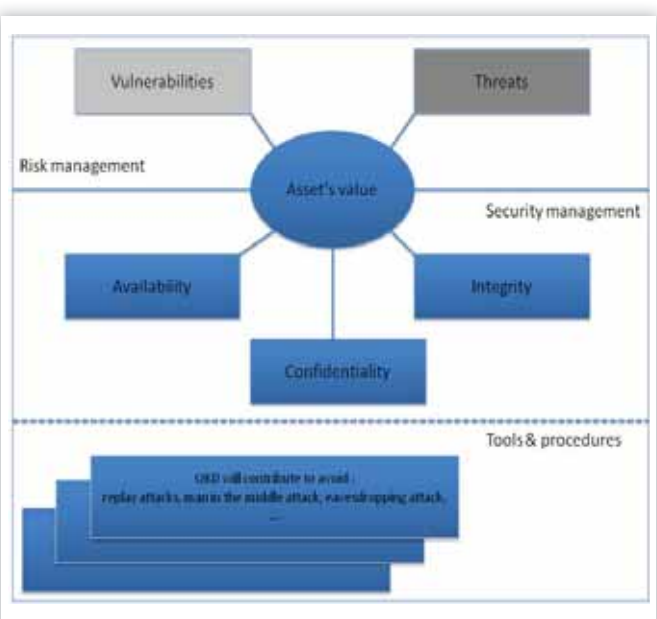


Figure 1. *Quantum key distribution (QKD) in the production chain of information security*

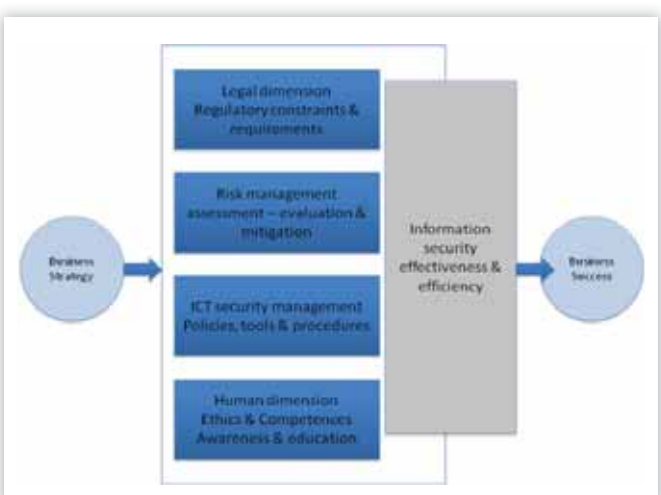


Figure 2. *Fundamental information security fundamental components to achieve business success*

risk without specifying them in any detail. These regulations do impact, directly or indirectly, the way an information system is managed and how information security is operated.

As was seen in the responses to the introduction to SOX, for example, some organisations consider the need for compliance as being a catalyst for resolving long-overlooked security problems, but at the same time the need to satisfy regulatory requirements increases the complexity level of the information security management process.

For some organisations, QKD could be seen as part of an alternative solution for demanding users. It could contribute to enhance their competitive and reputation advantage by ensuring long-term confidentiality and helping them to be complaint with relevant regulations. QKD can be reasonably combined with other security techniques to contribute to the security of information infrastructures. As will be described in Section 3 below, in the near future QKD could be integrated in native mode into backbone infrastructures as a component of an improved security landscape which, in the right circumstances and implemented in the right way, could prove to be a cost-effective and efficient contribution towards improved security.

From experimental proposition to market solutions

Since the first practical demonstration of quantum key distribution (QKD) over a distance of few centimetres performed by Bennet et al in 1993, research and experimental technology has demonstrated ample progress, so that significantly increased key rates and distances can be achieved with contemporary systems. Today, quantum key distribution is no longer confined to large optical tables in laboratories; systems are available and capable of automated continuous operation using readily available standard telecom fibres. There is a shift of interest in QKD systems and their scope for usability nowadays covers not only the areas of physics applications but also questions of interest to ICT security managers (Figure 3).

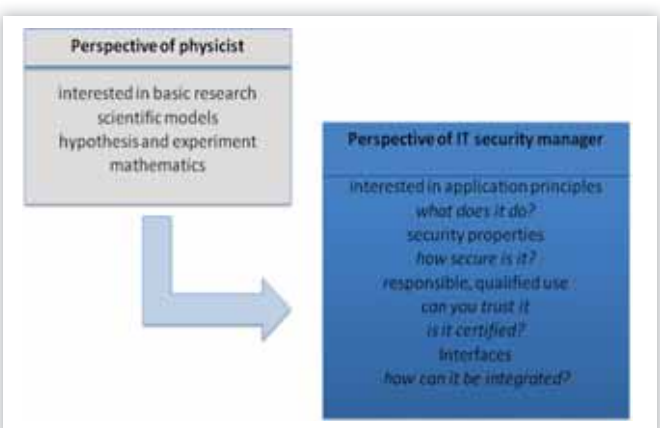


Figure 3. *A shift in QKD perspectives.*

The practical application of QKD technology for securing digital communication has already been demonstrated in several experiments and exhibitions over the last decade [1,2,3]. Yet examples of actual deployment of QKD in a commercial setting are very scarce. One pioneering real world application is the secure link between a ballot counting centre and a government data centre in Geneva [4], which has been in service during several elections, starting with parliamentary elections in 2007. Another production application is located in Durban, South Africa, where several links in a metropolitan area backbone are secured on a daily basis using QKD keys [5]. There are also indications of more systems being deployed for testing and also regular operation in financial institutions and defence data networks – a sector which usually does not publish details of its security systems

The first attempt to validate the practical applicability of quantum key distribution was carried out in the four-year project SECOQC (*Development of a Global Network for Secure Communication based on Quantum Cryptography*) of the 6th Framework Programme of the European Community. Six technologically different systems were operated under realistic assumptions in a quantum key distribution network in Vienna in autumn 2008, feeding user level applications with highly secure cryptographic keys. This world premiere attracted worldwide attention [2].

Although QKD systems today appear mature compared to the first experimental realizations, more technical improvement is required before a wide scale real-world deployment for qualified use can be considered. Moreover, QKD systems need to be compatible with existing interfaces for handling cryptographic keys. They need to be compatible with system and service management procedures within ICT infrastructures.

As regards functionality, QKD can be regarded as so-called cryptographic. Cryptographic primitives are low-level building blocks for implementing cryptographic systems to offer security services, such as, for example:

- encryption (confidentiality);
- authentication (integrity, proof of origin);
- key distribution;
- digital signature scheme (proof of origin, integrity, non repudiation);
- other primitives for commitment schemes, oblivious transfer, etc.

In practical applications, QKD usually delivers the cryptographic keys subsequently used by other crypto primitives. It should be noted here that the secure combination of cryptographic primitives (and the composition of cryptographic protocols) is an issue that has to be evaluated carefully.It shall only be hinted here that the overall security of a cryptographic system is usually determined by the security of its weakest link.

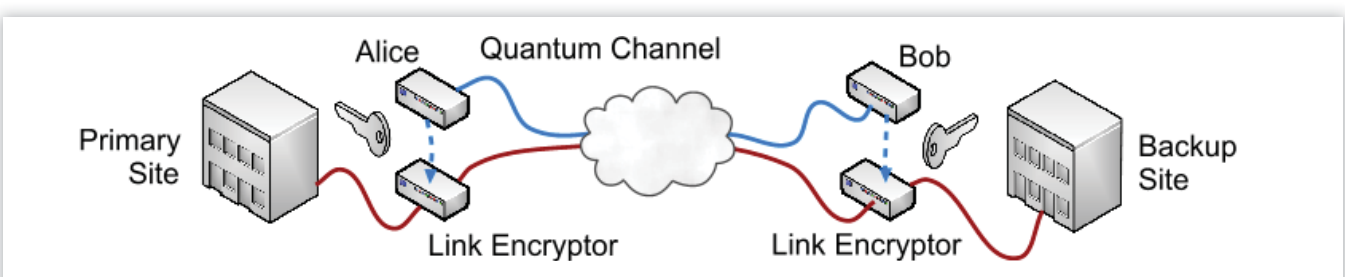


Figure 4. *A data link layer use of QKD to enhance security of backup operations*

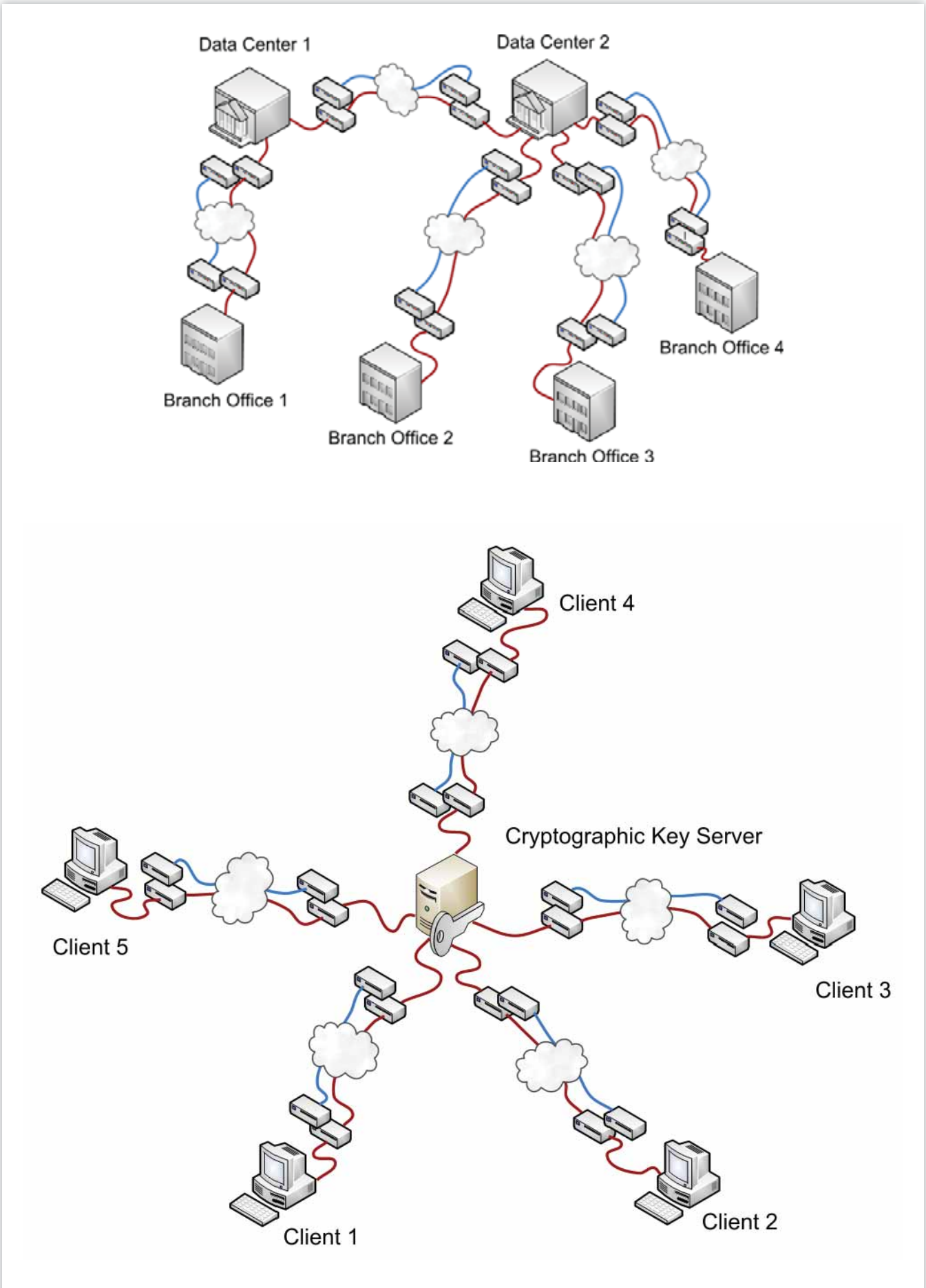


Figure 5. *Examples of QKD uses for enterprise metropolitan area network and for securing key server.*

The integration of QKD can be carried out for example directly into high-level applications, into transport layer TLS/SSL enabled with QKD session keys, integrated in the Network Layer within IPSec/IKE (which can use a QKD subsystem for security associations), or at the Data Link Layer within the Point to Point Protocol and its variants [12, 13, 14].

QKD integrated into the Data Link layer can satisfy security needs in relation to offsite backup and business continuity (Figure 4).

Integrated at the Transport or Network level, QKD can help resolving the problem of the protection of key distribution channels between data centres and their clients for authentication purposes (Figure 5 (a)), or to protect the channels over which encryption systems (clients) that consume cryptographic keys access a cryptographic key server that creates and manages keys (figure 5 (b)), for example.

When we look at potential customers of QKD, there could be multiple stages of customers in a typical supply chain for QKD, ranging from vendors to integrators, service providers, and end users. We have identified a group of *owners* of QKD systems and communication infrastructures (i.e., fibres), among them governments, financial entities, communication service providers, critical infrastructure providers, and military agencies. These owners between them manage all the hardware and software.

Another group is the so-called end users who buy a service from a service provider. Examples are enterprises of all sizes, private individuals, and possibly also entities from the *owners* group subscribing to a security service. A third group of potential customers includes the *community*, unions of member states, and governments who strive for the ultimate goal of providing reliable and resilient information infrastructures for themselves to enable more effective inter- and intra-governmental communications. In such a *new generation Internet*, QKD can have its specific areas of application, side by side with other cryptographic techniques and primitives [10, 11].

In this context QKD can be used for security services between the nodes of a backbone network to enhance the security level, especially the authentication of the links between the nodes, such that no unauthorized nodes can be inserted undetected. Otherwise, QKD may be used to provide a resource that can be used as a service by the carrier/network operator. Essentially QKD could be seen as a point of departure for changing security paradigms: as small challenges in the overall process are met by the application of such technologies, resources can be directed to newer and wider strategic challenges.

Moreover, last mile Passive Optical Networks (PONs) need encryption because the entire downstream can be *seen* by all endpoints, or optical network units (ONUs). As PONs are also transparent for quantum information QKD can be useful for providing communication security and be implemented in an 'asymmetrical setup' (one source, many detectors, or the other way around) (Figure 6).

We can also mention that QKD can be used in free space communication (long haul service) to facilitate highly secure key distribution between far remote sites without specific trust assumptions about intermediary nodes.

These examples of usage cases for QKD applications should be developed in further depth and lead to showcasing its added value for security systems in order to foster market creation and convince a wider public of their usability for next generation networks.

The added value of standardization

The qualified practical use of QKD requires that its users trust QKD systems, which is usually achieved through a complex assurance procedure including security specification, evaluation, and certification according to a standardized methodology. An element that is specifically required for the security certification of QKD systems is a framework for the underlying theoretical proofs of information security, which again requires standardized properties of optical components, like photon sources and detectors.

QKD Standardization would contribute to shifting from a technical innovation to market solution by enabling the dependable practical application of QKD as part of security solutions through the development of standards for interfaces, as well as for the qualification of QKD system components and the security certification of entire QKD systems.

Practical applications would be supported by a security certification scheme for QKD systems and the development of a reference model for business application [7]. The latter activity includes the development of use cases for the practical and commercial application of QKD systems with two main goals: to identify and advertise possible areas of application for QKD and to derive specific requirements for QKD systems [8].

Of course, the development of the tools needed for reliable certification and the achievement of an appropriate level of maturity in the use of this technology is a process that will not be completed overnight. As with all technological advances, the wider adoption of QKD will require that the lead is taken by a small number of pioneering organisations who are prepared

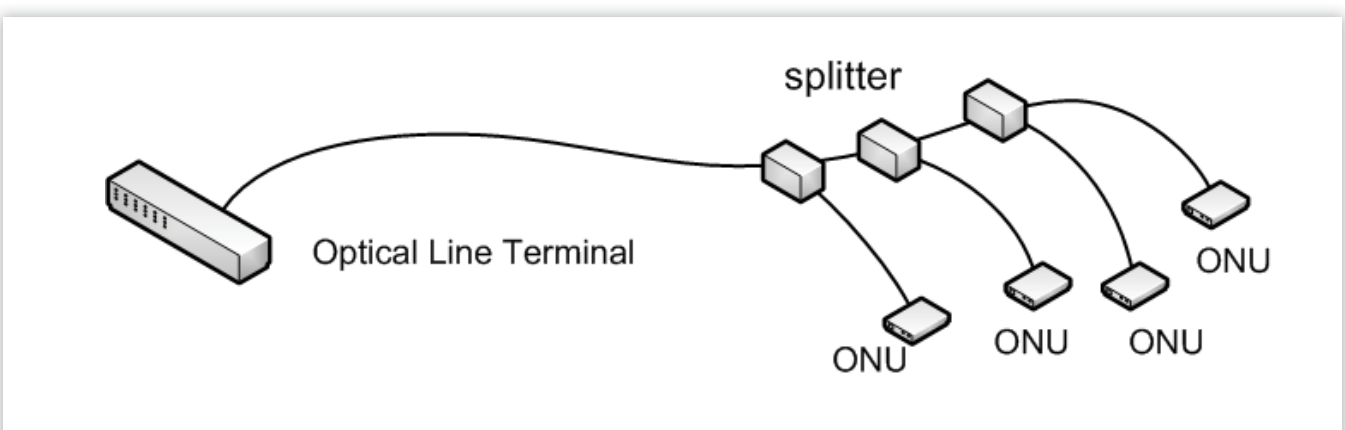


Figure 6. *QKD in Passive Optical Networks for High Security Access Network.*

to invest on the basis of the potential of the technology and share the practical results of their efforts with the research community.

Open issues and perspectives

Being a new technology, at the moment QKD naturally shows some limits that are the subjects of an ongoing process of improvement. Implementation problems are still being addressed and vulnerabilities tend to arise more from the hardware side than the software side. In any case, the robustness provided by applying a law of quantum physics to enhance a secret key distribution process is of no use if the hardware and computational implementation are fallible and can be cracked. Today most quantum hacking exploits holes in hardware implementation. Quantum cryptography technology has not been intensively tested or attacked or at the same time validated by a large number of experts, and thus experts are currently unable to point to a long history of effective and reliable security based on the implementation of these technologies.

Another issue of intense discussion (and widespread misconception) is the security of QKD. Misleading claims of *unbreakable cryptography'* paired with sensational reports on *quantum cryptography broken*, have been the source of persistent confusion regarding this important issue, not only among prospective customers and users but also among the scientists developing this technology.

The theoretically achievable security level of QKD is fundamentally different (and not only *higher*) than with other key distribution methods. Specifically, the security of QKD can by principle be made arbitrarily high and is not based computational assumptions (i.e., that the attackers do not have very powerful (quantum or grid) computers at their disposal). But, in practice, a specific QKD implementation, like any other ICT security system implementation, may exhibit unintentional side channels, which, when exploited, have the potential to entirely subvert its security. QKD systems even have additional attack vectors related to their optical subsystem. Regarding side channels, QKD is fundamentally equal to other key distribution methods: the theoretical security of QKD does not prevent side channels. Several side channel attacks, mainly on the optical subsystem, have been practically demonstrated and published (e.g., [9]). The issue of side channels is well researched and although side channels cannot be principally ruled out, there are best practice strategies to deal with them during system design and system operation life cycle phases.

At the current time, the key exchange rates form a limit to performance, but the performance of single QKD links is increasing. Short distances are also a concern today but the distances are constantly growing and may be further extended thanks to the use of the network approach with trusted repeater stations. This is a very useful approach for metropolitan area size networks, which would be the initial area of application of QKD.

The implementation of a QKD network is expensive, like the implementation of any other innovation, but it does not create disruption costs as it can be deployed in parallel to existing key distribution channels. In practice implementation costs are principally concentrated on hardware or device related costs, which are not as expensive as the administrative costs of a service disruption or of a service upgrading to another technology. In all the cases, a QKD network offers a long-term service, its dedicated costs of implementing being easily redeemable. Working with a long-term vision places an organization in a higher competitive position than its competitors. Apart from the remaining long-term security operations, the project provides cost savings in terms of investments, upgrading costs, changing costs, risk related costs, etc.

The use of an inherently secure way to communicate should allow the confirmation that the organisation using such a technology is no longer at the same level of insecurity as its competitors. Of course, some costs are generated, but, in the event of a successful and demonstrably reliable and beneficial implementation, these costs could be balanced by taking into account its long-term use and the potential gain in prestige (in image, reputation and in terms of confidence); thus, these expenses could thus be considered as being justifiable and a worthwhile investment.

QKD can fulfil an organisation's primary objective, which is to have a better security without a significant level of added costs.

QKD network implementation currently requires dedicated fibre. With hundreds of fibre strands in contemporary cables, dedicated fibres are not a very costly problem. In spite of this, future development could allow the use of wavelength-division multiplexing (WDM). This is a technology for multiplexing multiple optical carrier signals on a single optical fibre by using different wavelengths (colours) of laser light to carry different signals. This allows a multiplication of capacity, in addition to making it possible to perform bidirectional communications over one strand of fibre.

Achieving confidentiality is one of the cornerstones of security measures. This is one of the objectives to be fulfilled by existing cryptographic implementations. The reliability and robustness of the cryptographic mechanisms essentially rely upon cryptographic keys (key generation, distribution and storage, key secrecy). With the increase of computational power, current encryption and decryption methods, based on secret keys that support secure communications, are under threat. The lack of key security in relation to classical encryption methods means that these technologies no longer ensure a high level of security.

Information has become a very important asset for today's organizations, which are more and more subject to regulatory compliance issues. Added to the fact that information security officers could be subject to legal pursuits in respect of non-compliance caused by a lack of ICT security means (civil and penal responsibilities), they have to rely upon strong technical security solutions. Quantum cryptography contributes to answering these needs.

Cryptographic solutions must support reliable and provable confidentiality services in order to support today's business competitiveness and effectiveness in an uncertain world. It has been demonstrated that if underlying cryptographic mechanisms are solely based on algebraic complexity, they are no longer sufficiently secure. The only possibility to bypass this fact is to change the mathematical cryptographic paradigms by integrating quantum theory into cryptographic solutions to create inherently secure mechanisms.

Rethinking fundamentals in cryptography should be a solution for developing a new vision of security for the performance of transactions that are critical for institutions and people.

It allows breaking the vicious circle that assumes that only an entity that offers commercial security solutions can really master the data confidentiality of an organisation. With quantum key distribution, institutions and people have, for the first time, the means to be sure that their data are under their own

control and cannot be obtained by eavesdroppers without the sender's or recipient's knowledge.

Adopting quantum random numbers generators is the first step towards enforcing actual cryptographic robustness in every day transactions. It could be done very easily and is cost effective. The second move towards high security is to transmit confidential data through point-to-point connections secured by the combination of quantum key distribution and strong classical encryption algorithms. This choice has already been made, for high value applications and long-term secure data retention, by leading institutions that are highly security aware, in order to obtain strong competitive advantages in the marketplace.

We believe that integrating QKD into network backbone in native mode should become a reality in the near future. Next generation networks will thus support QKD for critical applications and services. Insurance processes and audit security evaluations will consider this kind of implementation as a key strategic advantage.

Conclusion

Of course, cryptography in any form, and especially the specific element that is QKD, has only a small part to play in the management of security and the achievement of acceptable levels of security for an organisation. Security remains a question of the weakest link and even within the limited field of cryptography within a security environment the utilisation of QKD cannot by itself guarantee an increased level of security. It needs to be implemented correctly within an environment that itself is appropriately managed and configured.

QKD is thus not a solution to all the ills besetting risk and security managers and should not be marketed as such. We believe, however, that the theoretical and practical demonstrations of its use and potential, and constantly improving techniques for implementation and support, show that it will very soon have an important part to play in the resources available to security practitioners and that its costs of implementation will rapidly be compensated many times over by the improvements it brings to the practical control of security within infrastructures. We are confident that the research being undertaken by other quantum security groups will contribute to making this vision a reality.

References

[1] Elliott C et al. 2005 Current status of the DARPA quantum network (http://arxiv.org/abs/quant-ph/0503058)

[2] Peev et al. 2009 The SECOQC quantum key distribution network in Vienna, New Journal of Physics. 11 075001 (*http://iopscience.iop.org/1367-2630/11/7/075001*)

[3] Sasaki M et al. 2011 Field test of quantum key distribution in the Tokyo QKD Network Optics Express Vol. 19 Iss. 11, pp 10387-10409

[4] Quantum cryptography to protect Swiss election *http://www.newscientist.com/article/dn12786-quantum-cryptography-to-protect-swiss-election.html* (online 16.07.2010)

[5] Mirza A and Petruccione F, 2010, Realizing long-term quantum cryptography, J. Opt. Soc. Am. B 27, A185-A188 (online *http://www.opticsinfobase.org/josab/abstract.cfm? URI=josab-27-6-A185*)

[6] Ghernaouti-Hélie S, Tashi I, Länger T and Monyk C 2008 SECOQC Business White Paper Journal publication pending online 1.1.2009 (http://www.secoqc.net)

[7] Länger T and Lenhart G 2009 ETSI Standardization of quantum key distribution and the ETSI standardization initiative ISG-QKD, New J. of Phys. 11 055051 (*http://iopscience.iop.org/1367-2630/11/5/055051*)

[8] ETSI ISG-QKD Group Specification "QKD; Use Cases" Version 1 online *http://webapp.etsi.org/WorkProgram/Report_WorkItem.asp? WKI_ID=29096*

[9] Vakhitov A, Makarov V and Hjelme D R 2001 Large pulse attack as a method of conventional optical eavesdropping in quantum cryptography J. Mod. Opt. 48 2023-38

[10] Dodson D et al. 2009 Updating Quantum Cryptography Report ver. 1, arXiv:0905.4325. (*http://arxiv.org/abs/0905.4325*)

[11] Maurhart O and Lorünser T and Länger T and Pacher C and Peev M and Poppe A 2009 "Node modules and protocols for the Quantum-Back-Bone of a quantum-key-distribution network"; Presentation: 35th European Conference on Optical Communication - ECOC 2009, Wien; 20.09.2009 - 24.09.2009; in: "Optical Communication" IEEE ISBN: 978-1-4244-5096-1; 2 S.

[12] Sfaxi M A2007 *Improving telecommunication security level by integrating quantum key distribution in communication protocols*; PhD Thesis, University of Lausanne

[13] Thi M , Sfaxi M A, Ghernaouti-Hélie S 2006 802.11i Encryption Key Distribution Using Quantum Cryptography Journal of Network, Volume 1, number 5. Pages 9-20

[14] Peev M et al. 2009 The SECOQC quantum key distribution network in Vienna New J. of Phys. 11 075001 (37pp)

SOLANGE GHERNAOUTI-HÉLIE

Solange Ghernaouti-Hélie is a professor in the Faculty of Business and Economics at the University of Lausanne – Switzerland; where she founded the Swiss Cybersecurity Advisory and Research Group, which deals with the socio-economic, managerial, legal, and technological dimensions of information and communication technology security. In 2011, she was named by l'Hebdo magazine as one of the 100 most important personalities in French-speaking Switzerland and by Bilan magazine as one of the 300 most influential figures in Switzerland. She is an emblematic figure among scientists on the cutting edge of cybersecurtiy's research field.

She is an active independent security consultant and an influential analyst on cyber security, cyber crime and cyber warfare related issues, possessing extensive experience of information security governance, cyber security strategies, on the evaluation of security policies of cyber threats and cyber risks. She was a key researcher on the integrated European Research project known as SECOQC (Secure Communication based on Quantum Cryptography, 2004-2008). She has authored more than twenty books on telecommunications and security issues, including "Information Security Evaluation – a Holistic Approach" (with Dr. I. Tashi) EPFL Press 2011 and "A global treaty on cybersecurity and cybercrime: a contribution for peace, justice and security in cyberspace, Second edition, 2011 (with Judge S. Schjolberg Cybercrimedata), Oslo 2011.

www.hec.unil.ch/sgh/

THOMAS LÄNGER

DI Thomas Länger is computer scientist of Technical University Vienna. His main areas of expertise are in the field of information technology security with an emphasis on security assessment and certification according to ISO/EN 15408 "Common Criteria". In 2000 he joined the Austrian Institute of Technology (AIT), starting in the Safety and Security department in the Security Certification Lab. In 2003 he changed to the newly formed Quantum Technologies Division. He was responsible for the design and essential parts of the implementation of the certification subproject of the Integrated Project SECOQC of the 6th framework programme. Since then, he designed and managed several projects with a focus on certification and practical application of Quantum Key Distribution. Currently, he is involved in a quantum metrology project, and leads a market study for the application of quantum communication in satellite communications networks. Thomas Länger has been founding chairman of the ETSI Industry Specification Group for Quantum Key Distribution (ETSI ISG-QKD) since 2008.

SECURING YOUR VITAL COMMUNICATIONS

PAUL BAKER

Almost every application written today uses network communication services to transfer data. Most of these transfers are performed over insecure and untrusted networks, such as the Internet. We would prefer to make sure that we can transfer data without somebody else eavesdropping on it.

A good standard for securing communications exists in the form of Secure Socket Layer (SSL) and its successor Transport Layer Security (TLS). But it's easier said than done to implement a secure channel in your application, especially in case you are not working on a standard PC platform but on an embedded or mobile platform. This article will show you how to add secure channels (and basic cryptography) to your application in a portable, light-weight and readable fashion.

Introduction

This piece focuses on how you can use the small PolarSSL library to add SSL/TLS secured channels to your existing network application, written in C, without much fuss. You will learn the basics about SSL/TLS communication and about integrating it into your application. At the end you will be able to add SSL/TLS to applications whenever you need it and you'll have learned a simple though much-used alternative to the complex library OpenSSL.

This article expects readers to have a basic understanding of network programming and cryptography. Yet no in-depth knowledge about specifics of cryptographic building blocks, such as the internals of AES, RSA or SHA-256 is required to understand and perform these changes.

Body

SSL/TLS

SSL/TLS is defined in a number of RFC's and has been updated over the past years from SSL 3.0 to TLS 1.0 (RFC 2246), TLS 1.1 (RFC 4346) and now finally to TLS 1.2 (RFC 5246). You still see a lot of applications and servers that use TLS 1.0 and TLS 1.1 around, as TLS 1.2 is not yet widely used. SSL/TLS is backwards compatible, as such this is often not a real issue. Only if a client or server has explicitly specified that it will only accept connections of a specific version, an issue arises if the other side does not support it.

When a client has set up a connection with a server then SSL/TLS starts with a handshake phase. In this handshake phase, the client and the server decide on the important aspects of the connection, such as the verification of the identity of both sides, the cryptographic algorithms used to secure the connection and the actual key to be used. The combination of cryptographic algorithms used to secure the channel, is called a ciphersuite in SSL/TLS. A ciphersuite is a combination of a key-exchange algorithm (such as RSA, Diffie-Hellman, ECDH), an encryption algorithm (such as RC4, AES, CAMELLIA, 3DES, DES) and a message authentication algorithm (such as MD5, SHA1, SHA-256). Picking the right ciphersuite for the job can be tricky, but the default suite is in most cases a safe bet (RSA, AES and SHA1).

In order to get to that point, both the server and the client have to reach agreement upon which ciphersuite and secret key they communicate with; without anybody else learning about the latter. Within this phase there is a fixed order and number of handshake messages that both parties can send. Some are required and provide the main flow of the negotiation and some are optional, depending on the availability of client and/or server certificates for authentication.

Basically the client suggests which ciphersuites it wants to use, but in the end the server decides which one of those is actually used. In case the client and the server have no common ciphersuites they agree upon, the connection will not be set-up.

In the end, the negotiation results in both sides having a selected ciphersuite and a secret key. And if any of the parties wanted to verify the identity of the other side, this has happened.

After this agreement all communication on the connection

between both parties is encrypted and authenticated with the established parameters. SSL/TLS provides a transparent layer to the application where it can send its data to, which then gets encrypted, authenticated and sent to the other side.

PolarSSL

The PolarSSL library serves as the basis for this article. Why, do you ask? Isn't OpenSSL the de-facto standard? Yes, it often is, but have you ever asked anybody if they understood what happened underneath, if they could find how to do non-standard things in the documentation and if they were happy with their code afterwards? OpenSSL is an excellent library that can do nearly anything, but one thing it's not is small and easy. I may be biased as I'm also lead maintainer for the project, but outside developers using PolarSSL, are often impressed by the ease of use.

The PolarSSL library has been designed to easily integrate with existing (embedded or regular) applications and to provide the building blocks for secure communication, cryptography and key management. PolarSSL is easy to understand and the code is readable, which is sort of unique in the SSL/TLS and cryptographic world.

PolarSSL is designed to be as loosely coupled as possible, allowing you to only integrate the parts you need without having overhead from the rest. This also results in a very low memory footprint and build footprint for the PolarSSL library. By eliminating parts you don't require in your system you can get build sizes from as low as 30 kB to a more typical 110 kB for more fully featured setups.

PolarSSL has been written in the portable C language with embedded environments as a main target and runs on targets as broad as embedded platforms like ARM and AVR to PCs and Android phones, iPads, iPhones and even the XBox.

More important is the fact that large open source projects like PowerDNS1 and OpenVPN2 use PolarSSL as their cryptographic or SSL/TLS building block. And just recently the Dutch government gave their approval to use OpenVPN in combination with PolarSSL for setting up restricted VPNs.

Application stack

From the perspective of the application, it's useful to understand where SSL/TLS lives inside the network stack. Let's start with showing the major components that are involved.
In Figure 1 you see from the bottom up:

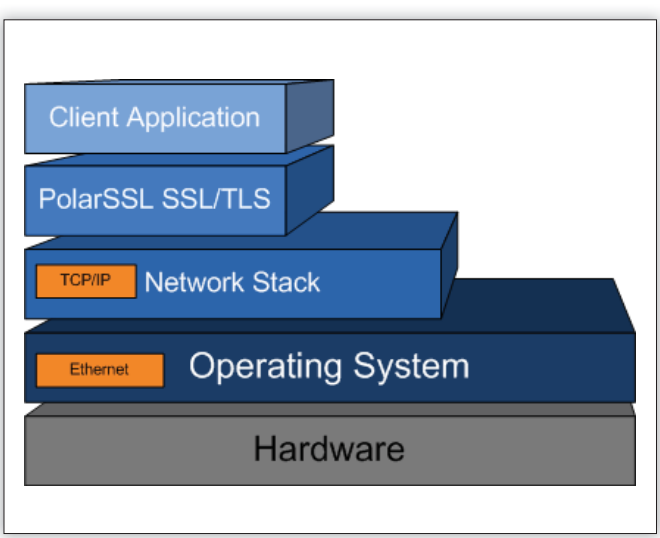


Figure 1. *Application stack for SSL/TLS*

- **Hardware**
  The hardware platform provides the physical processor, storage, memory and network interface.
- **Operating System**
  The Operating System provides the Ethernet driver and standard services. Depending on the OS, this includes scheduling, thread-safety and a full network stack.
- **Network Stack**
  Depending on the Operating System, the Network Stack is either fully integrated or is a separate module that provides an abstraction layer from the network interface. Most used external modules are the lwIP TCP/IP stack and the uIP TCP/IP stack.
- **PolarSSL SSL/TLS Library**
  Building on top of the network interface, PolarSSL provides an abstraction layer for secure communication.
- **Client Application**
  The Client application uses PolarSSL to abstract the secure communication from itself.

The precise steps to integrate PolarSSL in your application are very dependent on the specific components used above. In this article we will assume a regular Operating System, like Linux, or Windows with integrated BSD-like TCP/IP stack.

SSL/TLS integration

The most important PolarSSL module for this article is the SSL/TLS module that provides the means to set-up and communicate over a secure communication channel using SSL/TLS.

In general, the order of items to do are:

- Initialize an SSL/TLS context.
- Perform an SSL/TLS handshake.
- Send/receive data.
- Notify a peer that a connection is being closed.

In order to perform its function correctly the SSL/TLS module needs to be configured to understand the current situation. Many aspects of such a channel are set through parameters and callback functions:

- The endpoint role: client or server.
- The authentication mode: whether certificate verification for the client or server or both should take place.
- The host-to-host communication channel: send and receive functions.
- The random number generator (RNG) function to use.
- The ciphersuites that are used.
- A certificate verification function.
- Session control: session get and set functions.
- X.509 parameters for certificate-handling and key exchange.

PolarSSL can be used to create an SSL/TLS server and client as it provides a framework to setup and communicate through an SSL/TLS communication channel. The SSL/TLS part relies directly on the certificate parsing, symmetric and asymmetric encryption and hashing modules of the library. No external dependencies are required.

Example Client

So let's get down to business. We've talked a bit about the theory behind SSL/TLS and about PolarSSL. But the proof is in the pudding.

Listing 1. *A simple networking client application*

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <netdb.h>

#define SERVER_PORT 80
#define SERVER_NAME "localhost"
#define GET_REQUEST "GET / HTTP/1.0\r\n\r\n"

int main( void )
{
    int ret, len, server_fd;
    unsigned char buf[1024];
    struct sockaddr_in server_addr;
    struct hostent *server_host;

    /*
     * Start the connection
     */
    printf( "\n  . Connecting to tcp/%s/%4d...",
        SERVER_NAME,

        SERVER_PORT );
    fflush( stdout );

    if( ( server_host = gethostbyname( SERVER_NAME ) )
         == NULL )
    {
        printf( " failed\n  ! gethostbyname failed\n\n");
        goto exit;
    }

    if( ( server_fd = socket( AF_INET, SOCK_STREAM,
                      IPPROTO_IP) ) < 0 )
    {
        printf( " failed\n  ! socket returned %d\n\n",
                      server_fd );
        goto exit;
    }

    memcpy( (void *) &server_addr.sin_addr,
            (void *) server_host->h_addr,
                     server_host->h_length );

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons( SERVER_PORT );

    if( ( ret = connect( server_fd, (struct sockaddr *)
                      &server_addr,
                        sizeof( server_addr ) ) ) < 0 )
    {
        printf( " failed\n  ! connect returned %d\n\n",
                      ret );
        goto exit;
    }

    printf( " ok\n" );

    /*
     * Write the GET request
     */
    printf( "  > Write to server:" );
    fflush( stdout );

    len = sprintf( (char *) buf, GET_REQUEST );

    while( ( ret = write( server_fd, buf, len ) ) <= 0 )
    {
        if( ret != 0 )
        {
            printf( " failed\n  !
                write returned %d\n\n", ret );
            goto exit;
        }
    }

    len = ret;
    printf( " %d bytes written\n\n%s", len, (char *) buf );

    /*
     * Read the HTTP response
     */
    printf( "  < Read from server:" );
    fflush( stdout );
    do
    {
        len = sizeof( buf ) - 1;
        memset( buf, 0, sizeof( buf ) );
        ret = read( server_fd, buf, len );

        if( ret <= 0 )
        {
            printf( "failed\n  !
               ssl_read returned %d\n\n", ret );
            break;
        }

        len = ret;
        printf( " %d bytes read\n\n%s",
           len, (char *) buf );
    }
    while( 1 );

exit:
    close( server_fd );
    return( ret );
}
```

Let's assume we have a simple network client that tries to open a connection to an HTTP server and read the default page. That application would probably look something like Listing 1.

This is a very simple networked client application on a Linux operating system that does nothing more than set up all the network connectivity, opening a connection on port 80 (HTTP) to a server. In this case the client has localhost hardcoded as the server it connects to. After a connection is established, the client writes out a very basic HTTP GET request for the main page on the server and reads the result until nothing more is sent by the server. You probably can't make it more simple than this. Then again, it does not need to be more complex, as all important aspects that you have in your own application are here as well.

Adding Secure Communication

So now our task is to make sure that this simple HTTP client application can talk to a more secure HTTPS server. Adding SSL/TLS to an application requires a number of modifications. The main modifications are the set-up, configuration and tear-down of the SSL contexts and structures. Smaller changes are those made to the network functions for connecting to a server, reading and writing data.

Listing 2. *Additional headers for adding SSL/TLS*

```
#include "polarssl/net.h"
#include "polarssl/ssl.h"
#include "polarssl/entropy.h"
#include "polarssl/ctr_drbg.h"
```

Listing 3. *Variables and initialization of SSL/TLS*

```
entropy_context entropy;
 ctr_drbg_context ctr_drbg;
 ssl_context ssl;
 ssl_session ssn;
 char *pers = "ssl_example";
 entropy_init( &entropy );
 if( ( ret = ctr_drbg_init( &ctr_drbg, entropy_func, &entropy,
                            (unsigned char *) pers, strlen( pers ) ) ) != 0 )
 {
     printf( " failed\n  ! ctr_drbg_init returned %d\n", ret );
     goto exit;
 }
 memset( &ssn, 0, sizeof( ssl_session ) );
 memset( &ssl, 0, sizeof( ssl_context ) );
```

Listing 4. *Original code for setting up a network connection*

```
 if( ( server_host = gethostbyname( SERVER_NAME ) ) == NULL )
     goto exit;

 if( ( server_fd = socket( AF_INET, SOCK_STREAM, IPPROTO_IP) ) < 0 )
     goto exit;

 memcpy( (void *) &server_addr.sin_addr, (void *) server_host->h_addr,
                 server_host->h_length );

 server_addr.sin_family = AF_INET;
 server_addr.sin_port = htons( SERVER_PORT );

 if( ( ret = connect( server_fd, (struct sockaddr *) &server_addr,
                      sizeof( server_addr ) ) ) < 0 )
     goto exit;
```

Listing 5. *Setting up a SSL/TLS connection*

```
 if( ( ret = net_connect( &server_fd, SERVER_NAME,
                                      SERVER_PORT ) ) != 0 )
 {
     printf( " failed\n  ! net_connect returned %d\n\n", ret );
     goto exit;
 }
```

Setup

In order to set-up the SSL/TLS module we require a good random number generator and a SSL context and SSL session store. The random number generator is very important since this provides the basis for the secret key we wish to establish. For random number generation PolarSSL mainly uses the CTR_DRBG random number generator based on AES-256 in counter mode. The CTR_DRBG random generator is based on a NIST standard (NIST SP 800-90) and makes use of an entropy gatherer that uses operating system specific and generic pools of random to create the best possible random on a system.

To integrate this into the application we have to add some additional headers shown in Listing 2.

After we added the required headers, we can add the required variables and the initialization of the structures as shows in Listing 3. The personalization string pers is something that the CTR_DRBG random generator uses to be as unique as possible. It is therefore advised to create a unique string for your application code.

SSL Connection

In our generic TCP/IP client application (Figure 1), the application handles the socket() and connect() calls. But now we'd like PolarSSL to handle setting up the underlying connection. PolarSSL generally abstracts this inside its Network Layer (net.c). Thus the code in Listing 4. gets simplified as it's replaced by the code in Listing 5.

SSL/TLS Configuration

Now that the low level socket connection is up and running, we should configure the SSL/TLS layer. As described earlier, we have to configure how the SSL/TLS layer has to interact on the established connection.

Configuring the endpoint status of the module determines if the SSL/TLS layer will act as a server (*SSL_IS_SERVER*) or a client (*SSL_IS_CLIENT*).

The authentication mode for the module determines how strict the certificates that are presented are checked. In case we use *SSL_VERIFY_NONE*, this side does not check the certificate if it receives one. If we use *SSL_VERIFY_OPTIONAL*, this side will check the certificate if it receives one, but does not care if it does not. In case we use SSL_VERIFY_REQUIRED, we only continue with the connection if we receive and can verify the certificate.

The SSL/TLS layer needs to know which ciphersuites the application should accept for securing its connection. If the client specifies more than one, the server has the final say in which ciphersuite is used. By default only acceptably strong suites are enabled in the provided *ssl_default_ciphersuites* list.

To wrap things up, we have to set up the session cache as well.

Listing 6. *Configuring the SSL/TLS layer*

```
ssl_set_endpoint( &ssl, SSL_IS_CLIENT );
 ssl_set_authmode( &ssl, SSL_VERIFY_NONE );
 ssl_set_rng( &ssl, ctr_drbg_random, &ctr_drbg );
 ssl_set_dbg( &ssl, my_debug, stdout );
 ssl_set_bio( &ssl, net_recv, &server_fd,
                    net_send, &server_fd );
 ssl_set_ciphersuites( &ssl, ssl_default_ciphersuites );
 ssl_set_session( &ssl, 1, 600, &ssn );
```

With the code in Listing 6 we configure the module as a client that does not check the certificate it receives. In addition, it uses the initialized CTR_DRBG random generator, a simple debug callback function and we provide it with the input and output functions it needs to use for sending out network traffic. These functions (*net_recv and net_send*) are generic wrappers around the BSD TCP/IP stack. We default to the standard ciphersuite list and do a generic setup of the session cache.

Reading and writing data

After all configuration is done, we just need to make sure that our application talks to the network via the SSL/TLS layer. This is actually the easiest part of the entire process.

For writing to the network layer:

```
while( ( ret = write( server_fd, buf, len ) ) <= 0 )
```

becomes

```
while( ( ret = ssl_write( &ssl, buf, len ) ) <= 0 )
```

For reading from the network layer:

```
ret = read( server_fd, buf, len );
```

becomes

```
 ret = ssl_read( &ssl, buf, len );
```

Teardown

After the SSL/TLS connection closes from the other side, or if our application wants to exit, we need to cleanly tear down the connection and destroy any SSL/TLS related information.
So we need to replace the existing close function with the code from":

```
net_close( server_fd );
ssl_free( &ssl );
```

Further addition

With that final change, we are done. After changing SERVER_PORT to 443, compiling this application and linking it to the PolarSSL library, we now have an application that can talk basic HTTPS to a web server. The final code is also available as *ssl_client1.c* in the source code of the library itself.

But these are just the basics. If we want to make sure that both sides know they can trust each other, we need to add client and server certificates so that each side can verify the other. We did not dive into the matter of certificates and the often complex CA structures behind them in the introduction and we won't do so here. But let's look at the changes needed to integrate into the client the client certificate, its key and the CA certificate it trusts. We would need to add the variables to hold the certificates and the key, then we need to read in the certificates and the key and then tell the framework how to use them. The additional code required for Listing 3 and Listing 6 can be seen in Listing 7.

Of course we need to make sure that *crtfilename*, *keyfilename* and *cafilename* are properly set to the filenames we want to use for each respectively.

If we now change the parameter of *ssl_set_authmode()* in Listing 6 to *SSL_VERIFY_REQUIRED*, we require that we get a server certificate and trust it as well. A fully configurable version of the client application that handles the com-

mand line can be found as *ssl_client2.c* in the source code of the library.

If we want to convert a network connected server application as well, we need to make similar changes to that application. We will not dive into that in this article, but it suffices to say that the changes are comparable to the changes on the client side. A simple single-threaded example is available as *ssl_server.c* in the source code of the library.

Conclusion

In this concise article we have learned how to insert secure communications into a simple network connected client. PolarSSL was used and with only minimal changes (about +30 lines, -12 lines), we created a full-fledged SSL/TLS-enabled application. With the additional lines from Listing 8 we even added client certificates and required verification for the server certificate.

Of course there are situations where you are code-size or memory-constrained. In that case it's often beneficial to create a non-standard communication channel using only the basic cryptographic building blocks that PolarSSL can provide. But beware that this is a hard thing to do correctly. Cryptography is easy to do wrong and it's hard to see if that is the case. In the majority of cases using SSL/TLS is preferable.

I trust you now have the ability to add secure channels to your own code in the future and can prevent snooping on your important data channels. Good luck!

PAUL BAKKER

Paul Bakker(32) is a long-time tech-enthusiast with an interest in the security-arena, cryptography and entrepreneurship . His company Offspark advises government and commercial clients in the areas of cyber security, secure design, cryptography and secure development. He loves sharing his vision, providing advice and giving presentations to diverse groups.

Aside from dabbling in IT and security, Paul is passionate about helping startups and entrepreneurs. As an Angel investor he advises on issues ranging from technology choices to building traction and business models.

From 2002 to 2011 Paul had a full-time employment with the leading Dutch security company Fox-IT1. Paul led the Crypto & High Security department of Fox-IT for over 5 years, where as an active management member he was involved in military, government and general classified projects. In addition he was an active member in the development of numerous high-security products and solutions for the national security arena, including NATO and EU environments.

Listing 7. *Adding client certificate, key and trusted CA certificate to SSL/TLS client*

```
// Add in Listing 3
   //
   x509_cert cacert;
   x509_cert clicert;
   rsa_context rsa;

   memset( &cacert, 0, sizeof( x509_cert ) );
   memset( &clicert, 0, sizeof( x509_cert ) );
   memset( &rsa, 0, sizeof( rsa_context ) );

   if( ( ret = x509parse_crtfile( &clicert, crtfilename ) ) != 0 )
   {
       printf( " failed\n  !  x509parse_crtfile returned %d\n\n", ret );
       goto exit;
   }

   if( ( ret = x509parse_keyfile( &rsa, keyfilename, "" ) ) != 0 )
   {
       printf( " failed\n  !  x509parse_keyfile returned %d\n\n", ret );
       goto exit;
   }

   if( ( ret = x509parse_crtfile( &cacert, cafilename ) ) != 0 )
   {
       printf( " failed\n  !  x509parse_crtfile returned %d\n\n", ret );
       goto exit;
   }

   // Add in Listing 6
   //
   ssl_set_ca_chain( &ssl, &cacert, NULL, NULL );
   ssl_set_own_cert( &ssl, &clicert, &rsa );
```

A TOOLKIT FOR SAT-BASED CRYPTANALYSIS

PAWEŁ MORAWIECKI

In this article we would like to briefly describe a SAT-based attack – a method of attacking cryptographic primitives such as ciphers or hash functions. We also present key features of a toolkit developed by Paweł Morawiecki, Marian Srebrny and Mateusz Srebrny. The toolkit helps a cryptanalyst to mount the attack and automate some tedious work usually linked with this kind of attack.

SAT was the first known NP-complete problem, as proved by Stephen Cook in 1971. NP-complete problems is a class of problems for which we do not know an efficient algorithm which would provide a solution in reasonable time. In general finding a satisfying valuation for a given Boolean formula (SAT problem) belongs to this class, yet it turns out that many SAT instances can be solved surprisingly efficiently. They are solved by the algorithms called SAT solvers. Modern SAT solvers use highly tuned algorithms and data structures to quickly find a solution for a given formula, if possible. Many practical problems (for example, from industry) turned out to be easily solved by this method. In the last decade SAT solvers have been also successfully used as a tool in cryptanalysis. A general idea of SAT-based cryptanalysis is as follows. First, the problem (e.g., of finding a secret key in a cipher) is translated to a SAT instance in such a way that a satisfying valuation represents a solution to the problem. Then a SAT solver is run and we hope to obtain a solution in a reasonable time. Certainly, it is not always the case, it heavily depends on the problem hardness and how the formula is constructed.

A problems given to a SAT solver is usually encoded in Conjunctive Normal Form (CNF). In such CNF formula, each element is called a literal. A clause is a disjunction (or-ing) of literals. CNF is a conjunction (and-ing) of clauses. Hence, the constraints are presented to the SAT solver as an big *and* of *ors*. For example, (x v y v z) ∧ (x v !y) is a formula in CNF with three literals (!y denotes a negated form of y) and two clauses. CNFs describing ciphers or hash functions can consist thousands of literals and clauses.

One of the key steps in attacking cryptographic primitives with SAT solvers is a CNF formula generation. Such a formula completely describes the primitive (or a segment of the primitive) which is the target of the attack. Generating it is a non-trivial task and usually is very laborious. There are many ways to obtain the final CNF and the output results differ in the number of clauses, the average size of clauses and the number of literals. Recently we have developed a new toolkit called CryptLogVer which greatly simplifies the creation of a CNF. Here we describe only the main concepts. An interested reader can take a look at www.pawelmorawiecki.pl/cryptlogver where one can download a toolkit, find examples, tutorials and papers with the results.

Usually a cryptanalist needs to put a considerable effort into creating the final CNF. It involves writing a separate program dedicated only to a cryptographic primitive under consideration, some knowledge of logic synthesis algorithms and careful insight into the details of the primitive's operation. We have proposed the new toolkit which automates most of work while creating a CNF. Using the toolkit the complete process of a CNF generation has the following key steps:

1) Code the target cryptographic primitive in HDL;
2) Compile and synthesize the code in Quartus;
3) Generate boolean equations using Quartus inbuilt tool;
4) Convert generated equations to a CNF by our converter.

Steps 2, 3, and 4 are done automatically. The only effort a researcher has to put is to write a code in HDL. Normally programming and ‚thinking' in HDL is a bit different from typical high-level

languages like Java or C. However it is not the case here. For our needs, programming in HDL looks exactly the same as it would be done in high-level languages. In summary, the proposed toolkit can be especially helpful for researchers who start their work from scratch and do not want to spend too much time on writing thousands lines of code.

For the reader's convenience, we provide an example SystemVerilog code for SHA-1 used in the experiments with our toolkit. In many cases a code strongly resembles a pseudocode defining a given cryptographic algorithm. A reader familiar with C or Java should have no trouble adjusting the code to our toolkit's needs.

PAWEŁ MORAWIECKI

is an assistant professor at Kielce Univeristy of Commerce, Poland. In 2010 he gained his PhD degree from Warsaw University of Technology where he defended the thesis on logic synthesis. Currently his research focuses on cryptanalysis of modern ciphers and hash functions. He actively takes part in seminars in Institute of Computer Science of Polish Academy of Sciences.

Listing 1.

```
module sha1(IN, OUT);

   input [511:0] IN; // input here means 512-bit message block
   output [159:0] OUT;  // output here means 160-bit hash
   reg [159:0] OUT;


   reg [31:0] W_words [95:0]; // registers for W words
   reg [31:0] h0 ,h1, h2, h3, h4;
   reg [31:0] a, b, c, d, e, f, k, temp, temp2;
   integer i;


  always @ (IN, OUT)
  begin


h0 = 32'h67452301; h1 = 32'hEFCDAB89;
h2 = 32'h98BADCFE; h3 = 32'h10325476;
h4 = 32'hC3D2E1F0;


a = h0; b = h1; c = h2; d = h3; e = h4;


 W_words[15] = IN[31:0];  W_words[14] = IN[63:32];
 W_words[13] = IN[95:64]; W_words[12] = IN[127:96];
 W_words[11] = IN[159:128]; W_words[10] = IN[191:160];
 W_words[9] = IN[223:192];  W_words[8] = IN[255:224];
 W_words[7] = IN[287:256];  W_words[6] = IN[319:288];
 W_words[5] = IN[351:320];  W_words[4] = IN[383:352];
 W_words[3] = IN[415:384];  W_words[2] = IN[447:416];
 W_words[1] = IN[479:448];  W_words[0] = IN[511:480];


// extending W_words
  for (i=16; i<=79; i=i+1)
   begin
   W_words[i] = W_words[i-3] ^ W_words[i-8] ^ W_words[i-14] ^
                      W_words[i-16];
   W_words[i] = {W_words[i][30:0], W_words[i][31]}; //
                      leftrotate 1
   end


// main loop
 for (i=0; i<=79; i=i+1)
   begin


   if ((i>=0) && (i<=19))
      begin
      f = (b & c) | ((~b) & d);
      k = 32'h5A827999;
      end


   if ((i>=20) && (i<=39))
      begin
      f = b ^ c ^ d;
      k = 32'h6ED9EBA1;
      end


   if ((i>=40) && (i<=59))
      begin
      f = (b & c) | (b & d) | (c & d);
      k = 32'h8F1BBCDC;
      end


   if ((i>=60) && (i<=79))
      begin
      f = b ^ c ^ d;
      k = 32'hCA62C1D6;
      end


temp2 = {a[26:0], a[31:27]}; // a leftrotate 5


temp = temp2 + f + e + k + W_words[i];
e = d; d = c;
c = {b[1:0], b[31:2]}; // b leftrotate 30
b = a; a = temp;
end // end of main loop


h0 = h0 + a; h1 = h1 + b;
h2 = h2 + c; h3 = h3 + d;
h4 = h4 + e;

OUT = {h0, h1, h2, h3, h4}; //HASH
end
endmodule
```

INTERVIEW WITH VADIM MAKAROV

How did you get started in quantum computing and hacking?
I started in quantum information almost by accident. I got an offer of a PhD position in Norway 14 years ago, accepted it, and found the stuff interesting enough to keep doing it. At first we wanted to build improved quantum cryptography systems (faster and so on) like everybody else, but we quickly realised that trying to hack what the others build was much more exciting. Back then almost nobody did serious quantum hacking; now there are a few more researchers worldwide doing it.

What projects are you currently working on?
More hacking of quantum cryptography. Sorry, the details are secret. Results will be published in due time.

You are fairly well known for your key distribution attack in quantum cryptography. Could you describe your photon detector attack, how the equipment attacked is vulnerable and how the two parties don't detect delays in the key distribution?
In normal operation of quantum cryptography, the detectors in the receiver Bob are sensitive to single photons. This is critical for the security. We found that most detector types can be blinded by shining bright light at them. They stop seeing single photons, just as your eyes stop seeing stars
on the clear sky in daylight, even though all the stars are still there. The we found that if a still brighter flash of light is sent to Bob's detector (looks like a supernova to him), it mistakes it for a photon. With this bright light, the eavesdropper Eve controls Bob's detectors deterministically, the detection outcomes are no longer governed by the laws of quantum mechanics and the security vanishes.

There are no delays. Everything looks perfectly normal to the legitimate parties, while Eve has a full copy of the „secret" key they keep producing.

(Okay, there are a few more steps involved in the execution of the attack, but the above blinding of detectors is the core trick that allows it to succeed. If you want the gory details, read for example *http://arxiv.org/abs/1011.0105*).

Are there any countermeasures against your attack?
Plenty, but... most of the easy countermeasures are band-aid type, not a silver bullet. For example, it's easy to add a watchdog photodetector at the entrance of Bob, to watch for the bright light Eve sends. But exactly how much light would be too much to raise an alarm? And what if Eve blinds the watchdog detector too?

To state precisely, the fundamental difference between the classical public-key cryptography and the quantum key distribution is that the latter has a security proof. The security proof is a strict mathematical derivation that starts with the laws of physics and ends with a statement that if the observed bit error rate is below a certain threshold, then the key is secure. There are no other factors involved. The problem with the easy countermeasures is that they are not a part of this security proof, they become an extra unproven assumption. It is more difficult to invent a silver bullet countermeasure that is a part of the security proof, but we believe it is possible, and we are working on it now.

How many different types of commercial quantum systems are out there and how many are susceptible to your attack?
There are several small companies that advertise commercial quantum cryptography systems (I count at least four of them in different parts of the world). Also, several telecommunication giants have advanced prototypes in their industrial labs, ready to spring into action if the market grows. I know that the Swiss company ID Quantique routinely sells systems to customers: you can place an order and get it installed. I don't know about the other companies, haven't tried to order there recently.

Most of the systems on the market and in development could be vulnerable to our attack, but now the developers are well aware of it and have started patching. When we discovered our attack, we notified two companies privately some months in advance before the attack was made public. As one result, ID Quantique has deployed a patch to their customers some time ago.

Have any of the vendors started redesigning their detectors to prevent your attack? Have they consulted you on any mitigating measures?
Yes, several developers had to come up with countermeasures of varying efficiency. We don't think this is the end of the story yet, as the ultimate countermeasure has not yet been developed, as I explained above.

My group collaborates with IQ Quantique on the security issues. At this stage, the close cooperation helps both sides a lot.

Did you consult the vendors prior to publishing your work? What was their reaction?
Yes, of course we did. That was the only responsible course of action. The reaction was a mix of annoyance and appreciation. On the one hand, the vendors understand that independent scrutiny by hackers is necessary and is very beneficial to their

security products in the long run. On the other hand, we created an immediate problem demanding extra engineering resources, a problem they would rather not exist right now.

Given the types of consumers and limited suppliers, do you feel that manufacturers will ever threaten quantum hackers with lawsuits as many vendors have done over the past decade to other hackers in different fields of security work?

I hope no vendor will be silly enough. That would be the best advertising for my research lab, and the worst possible advertising for the vendor. Quantum cryptography is the highest-level security product, meant to withstand not just certain attack in a limited setting, but all attacks physically possible in principle. Supplementing its security by lawsuits would be pointless.

In fact, for the best security, I advocate a totally open quantum cryptography implementation, just as many classical cryptography implementations are today. But the companies are so far sceptical to this idea, due to intellectual property issues.

Are there any viable satellite links available? If so, how are they comparably and would they be vulnerable to your attack?

There have been demonstrations of quantum cryptography over terrestrial free air links, the longest over 144 km between two of the Canary Islands. No quantum satellite is flying, but there are competing plans by all major space agencies to launch a demonstrator, either as a stand-alone satellite or as an external pallet at the International Space Station. Notably China is among these: maybe they get their quantum satellite up first.

These links would in principle be just as vulnerable as their fiber counterparts. However, Eve gets an extra formidable problem: how to insert herself permanently into the light beam? Professor Zeilinger recently told me: „Vadim, you know, you can build an eight-meter, uh, no, fifteen-meter scaffold over that mountain ridge and get into the beam between our telescopes." That's the easiest I can think of; with a vertical beam to a moving satellite, Eve would be in a much much bigger trouble. But, nothing physically impossible.

Over the last two to three years what advances have you seen in quantum cryptography and what do you see coming in the near future?

I have seen working quantum networks demonstrated (last one in Tokyo area a year ago), I have seen quantum technology rapidly advancing to faster speeds and more practical equipment packages, and I have seen serious attention paid to the security at all layers very recently. I cannot predict the future, but the potential for wider deployment of quantum cryptography is there.

Questions by: Nick Baronian
Answered: Vadim Makarov

VADIM MAKAROV

is a research assistant professor at the Institute for Quantum Computing at the University of Waterloo, Canada. He also holds postdoctoral degree from Phoang University of Science and Technology in quantum optics and quantum information. He is the pioneer of in-band attacks by means of Quantum Cryptography. http://www.vad1.com/photo/family-archive/vm-20061224-19-4.jpg

E DETECTIVE

E - Detective®

Lawful Interception
Network Forensic Analysis
Internet Surveillance
Enterprise Information Security

Email
Chat
Web
FTP
P2P
VOIP
WebCam
HTTPS/SSL
TELNET

DECISION GROUP INC.

Address : 4/F No.31, Alley 4, Lane 36, Sec. 5, Ming-Shan East Rd, Taipei, Taiwan
Phone : +886 227665753 Fax : +886 227665702
Email : decision@decision.com.tw
Website : www.edecision4u.com

WE @ iViZ HATE FALSE POSITIVES

AS MUCH AS YOU DO

1st Cloud based Application Penetration Testing service

No Tools | No Consultants | Zero False Positives | Cost effective

www.ivizsecurity.com