

INVESTIGATING THE EFFICIENCY OF CRYPTOGRAPHIC ALGORITHMS IN ONLINE TRANSACTIONS

C. Lamprecht¹ A. van Moorsel P. Tomlinson N. Thomas

School of Computing Science,
University of Newcastle upon Tyne, UK

Abstract

Web Service security is an important factor for Web Services to gain increased acceptance. This paper presents how message level security is achieved in web services interactions and we evaluate a number of commonly used cryptographic algorithms to determine which are most suitable for the task. In particular we explore whether VeriSign's Trusted Services Integration Kit (TSIK) is a viable option for realising this. Furthermore, through measurement of TSIK as well as of an implementation using Java Cryptography Extensions (JCE), we conclude that TSIK provides an adequate level of security with minimal additional overheads. However, it would benefit from using SHA-256 and IDEA in future releases as well as decreasing algorithm operation time when processing larger messages.

1. Introduction

Since the advent of the World Wide Web, e-commerce has become a major source of interest for both businesses and customers. Since most transactions typically occur over the Internet, in the public domain, much research has been undertaken to use cryptographic algorithms to secure messages during these online transactions. Though emphasis has been primarily placed on the level of security which such algorithms provide, it is also of interest to evaluate their efficiency at doing so. This is particularly relevant in areas such as stock trading or online

bidding where the dynamic nature of the data accessed imposes real-time constraints on the transaction.

More recently Web Services have been met with growing interest from academia as well as industry due to its potential to provide a generic global service oriented network which is flexible enough to cater for individual service needs as well as providing increased interoperability between services. As such, e-commerce could benefit greatly from adopting Web Services technologies.

In this paper we will present three different cryptographic methods typically used to secure messages during online transactions. For each we will consider the available algorithms and conduct a comparative evaluation based on their performance. This will aid in determining suitable cryptographic algorithms for transactions with real-time constraints. We also consider Web Services' claim as a potential solution environment and evaluate Verisign's Trusted Services Integration Kit (TSIK), a hybrid solution based on a set of cryptographic algorithms, with respect to the level of security it provides as well as its efficiency at doing so.

We first discuss, in Section 2, what cryptographic methods are required to secure messages in online transactions and in Section 3 provide a comparative analysis of available algorithms using the Sun Java Cryptography Extensions (JCE) [SunJCE2005] as well as the Cryptix extensions for Java [Cryptix2005]. Section 4 details a comparative evaluation of TSIK's performance and level of security it provides with respect to using the Sun Java Cryptography Extensions. The paper concludes with a summary in Section 5.

¹ Communicating author,
C.J.Lamprecht@ncl.ac.uk

2. Securing Transactions

Online transactions typically require: message integrity to ensure messages are unaltered during transit; message confidentiality to ensure message content remain secret; non-repudiation to ensure that the sending party cannot deny sending the received message; and sender authentication to prove sender identity.

We provide a brief overview of the well-known cryptographic techniques available to achieve the above.

2.1 Symmetric cryptography

Symmetric cryptography tries to ensure message confidentiality by encrypting the message (the plaintext) using a secret key to produce an encrypted version of the message (the cipher text), which is then sent instead of the original message. Message integrity is implicitly provided, as altering the cipher text would result in an illegible decrypted message. 'Symmetric' refers to the fact that the same secret key is required to decrypt the message on the recipient's side. Typical symmetric encryption algorithms include DES, Triple DES, RC2, RC5, Twofish, Blowfish, IDEA and AES. Most symmetric algorithms can operate in two modes, namely Cipher Block Chaining Mode (CBC) or Electronic Codebook Mode (ECB). The former of which is considered more secure as it ensures that encrypting the same plaintext never produces the same cipher text. The main problem in this scheme is the key distribution problem; since the same secret key is used to decrypt the message, one must find a way to securely transport the key from sender to recipient.

2.2 Asymmetric cryptography (public key cryptography)

Asymmetric cryptography provides the same message security guarantees as symmetric cryptography, but additionally provides the non-repudiation guarantee. 'Asymmetric' refers to the fact that different keys are used for encryption and decryption. One key is kept secret ('secret key') and the other is made public ('public key'), and are both unique. The recipient's public key should be used during

the encryption process to ensure message confidentiality as only the recipient has the necessary secret key to decrypt the message. If, however, the message is encrypted using the sender's private key the sender cannot deny sending the message as his private key is unique and is only known to him. Typical asymmetric algorithms include RSA, ElGamal and DSA. Asymmetric cryptography is extremely powerful, but this comes at a cost. Especially for longer messages and keys, it is much slower than its symmetric cryptography counterparts [Adams2003]. This is due in part to the fact that, in order to achieve comparable security, asymmetric keys are generally around an order of magnitude longer than symmetric keys.

2.3 Hashing

Hashing tries to ensure message integrity by producing a condensed version of the message, known as the message digest, which is unique to that message. The hashing algorithm is publicly known and so the recipient can perform the same hash on the received message, to produce another message digest, and compare it to the received digest to assess whether the original message has been altered. Typical hashing algorithms include MD2, MD4, MD5, RIPEMD, SHA-1, SHA-256, SHA-384 and SHA-512. Hashing does not provide confidentiality, non-repudiation or authentication. On its own, hashing does not provide message integrity either as both the hash and the message could be replaced by a third party and so prevent the recipient from detecting the attack. Section 4.1 explains how hashing is utilized to ensure message integrity.

3. Cryptographic Techniques

The following section details a performance evaluation of the most common cryptographic algorithms for each cryptographic technique to determine their suitability for systems with real-time constraints. All experiments were conducted on a 1GHz machine with 256MB RAM running Linux Fedora Core. For each experiment a 1,137 byte plaintext file was used. All results for symmetric and asymmetric algorithms include key generation, algorithm

initialization and message encryption times. The experiments were repeated several times with negligible variance in the results.

3.1 Symmetric cryptography

This section details a comparative performance evaluation of a subset of symmetric encryption algorithms. Sun Java Cryptography Extensions [SunJCE2005] (referred to hereafter as *JCE*) as well as Java Cryptix Libraries [Cryptix2005] (referred to hereafter as *Cryptix*) are used for this purpose. Using Cryptix we furthermore investigate whether either Cipher Block Chaining Mode (CBC) or Electronic Codebook Mode (ECB) boasts a performance advantage. 128 bit key size was used for all algorithms with the exception of DES (56 bits), DESede (Triple DES using 112 bits) and Skipjack (80 bits) as they require fixed key sizes. Unless stated CBC mode was used.

3.1.1 Algorithms

Looking at Figures 1 and 2, the first observation to make is that there are significant differences between the observed durations shown in each graph; JCE took much longer than Cryptix for the same algorithm. The conclusion we draw from this is that the implementation has a large impact on the efficiency of the execution. This is further emphasized when individual algorithms are compared. Naively one would expect that Triple DES would take three times as long as DES. However, this is evidently not the case, being only about 25% slower in Cryptix and only very marginally slower in JCE. Clearly this is influenced by the implementation, and conceivably the Java Virtual Machine optimizations are also playing a part in apparently “speeding up” Triple DES.

The algorithm which consistently performed the best in our evaluation was IDEA. According to Schneier [Schneier1996], IDEA is approximately twice as fast as DES; in our experiments it was closer to three times as fast. Perhaps surprisingly, Blowfish was much slower, only a little better than DES and slower than algorithms such as Skipjack and Serpent. Blowfish was designed to be fast and requires

little memory [Schneier1996], but we did not find this Cryptix distribution particularly efficient in our experimental set up. AES (Rjindael) performed particularly badly in the JCE distribution, but less poorly in the Cryptix distribution. We were unable to satisfactorily explain this difference, except as further evidence of how the implementation of an algorithm can severely impact the actual performance.

What is not evident in these plots is the relative security of the different algorithms. In this respect key length is a good indicator, and so DES and Skipjack may be considered to be potentially less secure than others. Overall therefore it appears that IDEA is the best choice among the symmetric algorithms tested, as it provides adequate security as well as a fast execution time.

3.1.2 Encryption Mode

Figure 3 clearly indicates that neither mode shows a significant performance advantage. It would therefore seem prudent to use CBC mode during message encryption as discussed in Section 2.1.

3.2 Asymmetric cryptography (public key cryptography)

The results presented in Figure 4 were obtained using the standard Java Cryptography Extensions (JCE). The graph shows the average time to generate keys and encrypt 1,137 bytes of data. It can be seen that the RSA algorithm family outperforms that of Diffie-Hellman at all key lengths in our experiments. A key length of 1024 bits is currently considered to be the minimum secure length for both RSA and Diffie-Hellman. Diffie-Hellman with a key length of 1024 was also considered but yielded results 30 times slower than that of its RSA counterpart, and so is not shown for reasons of clarity. DSA also performed well but can only be used for non-repudiation purposes and not for data confidentiality. RSA and Diffie-Hellman are able to support both.

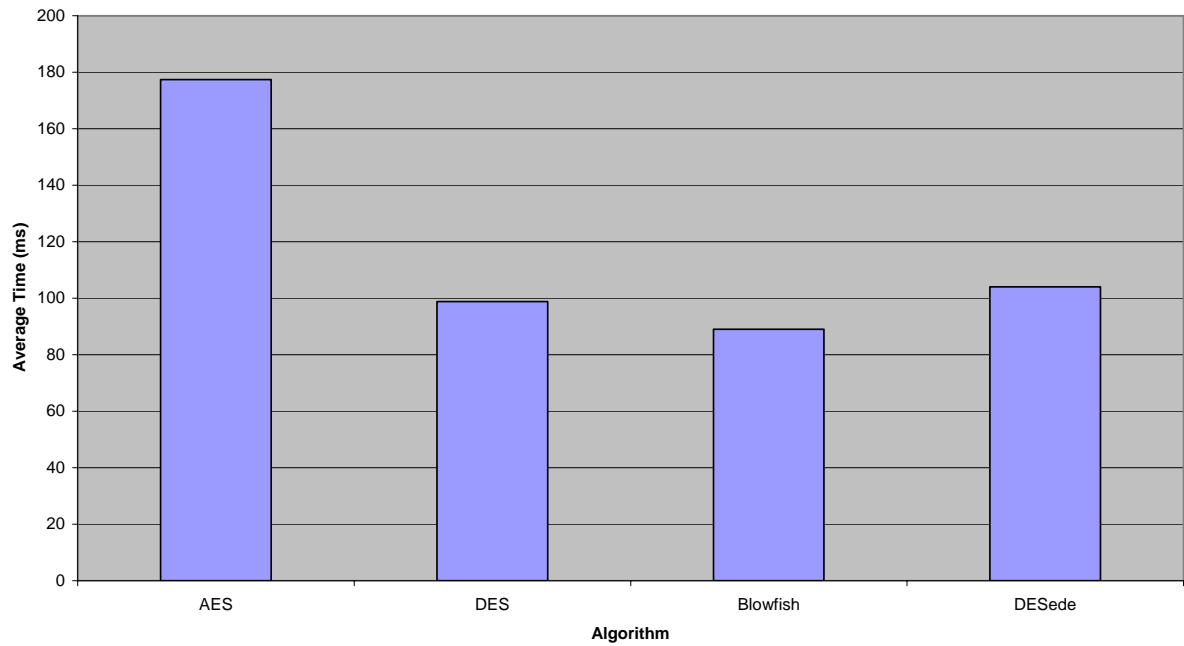


Figure 1: Average time to encrypt a 1137B file using JCE distributions

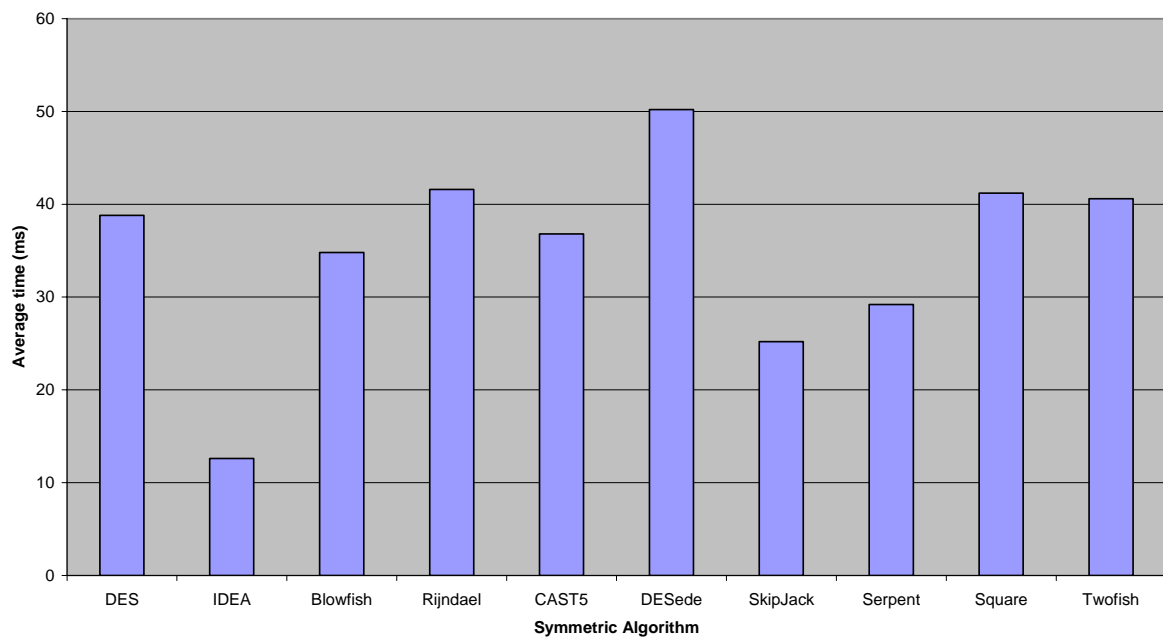


Figure 2: Average time to encrypt a 1137B file using Cryptix distributions

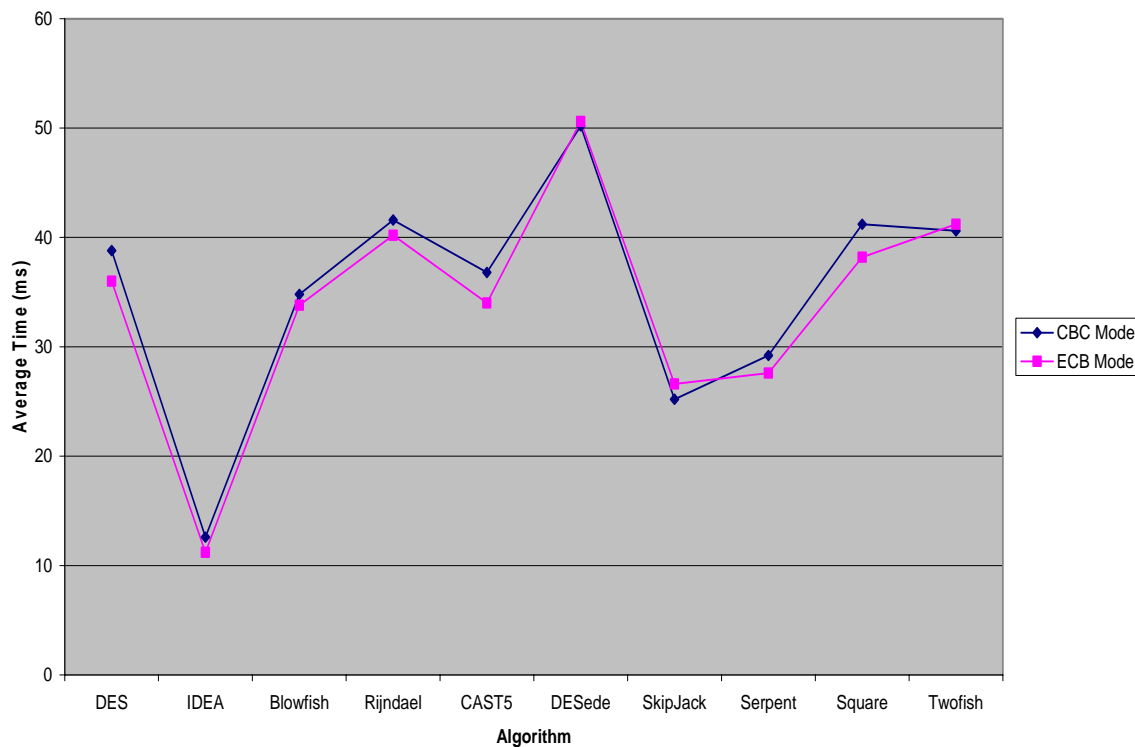


Figure 3: A comparison of symmetric algorithms operating in ECB mode and CBC mode

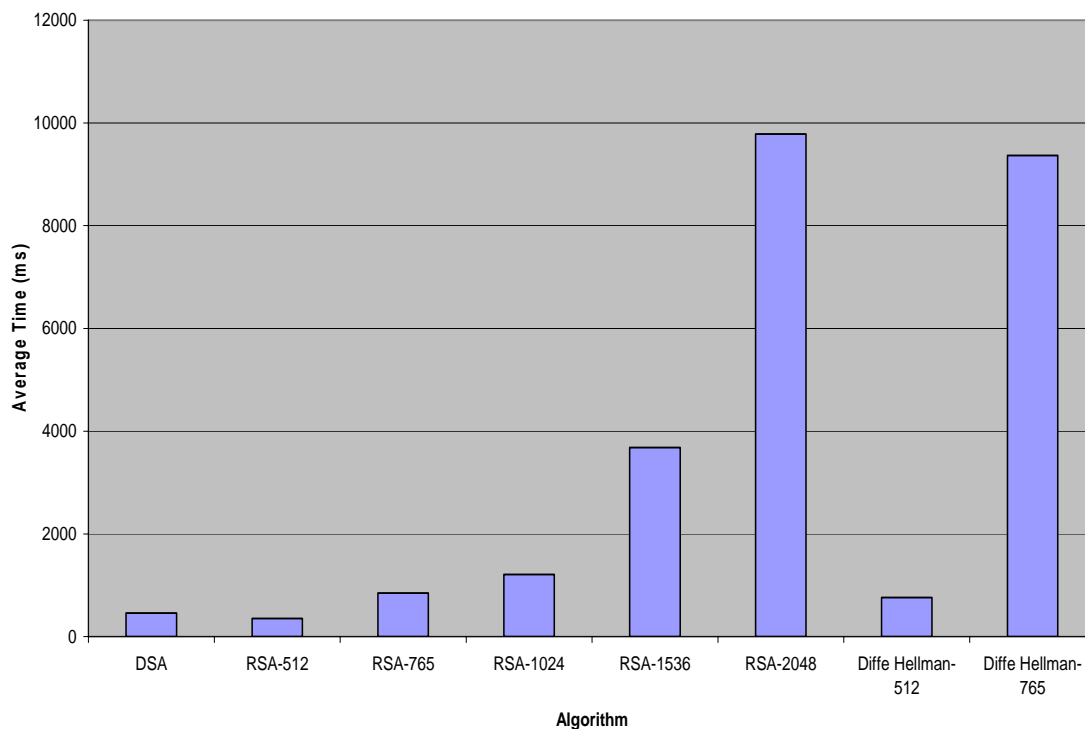


Figure 4: Average time for key generation and encryption using public key algorithms

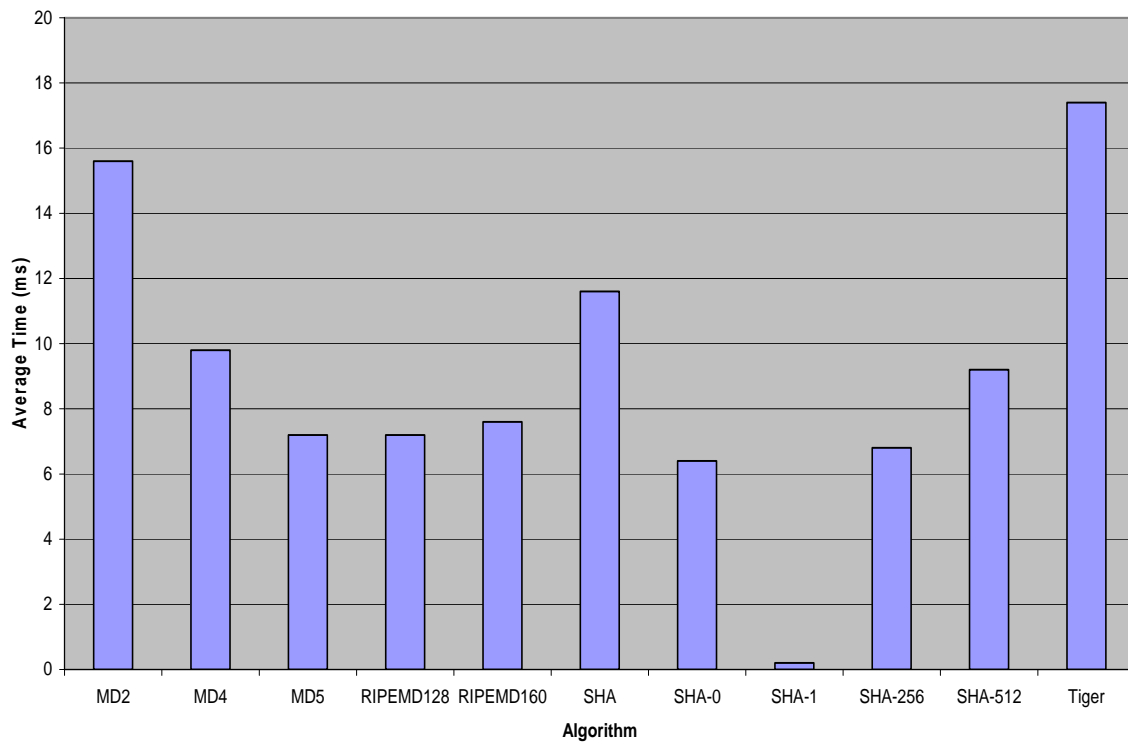


Figure 5: Average time to generate a message digest

3.3 Hashing

Java Cryptix Libraries were used in this experiment. 128 bit key size was used for all MD algorithms, 160 bits for SHA (unless otherwise stated) and 190 bits for Tiger.

As can be seen in Figure 5, SHA-1 significantly outperforms all other considered algorithms. Unfortunately SHA-1 has recently been shown to be less secure than initially anticipated and SHA-256 is currently recommended [Lenstra2005]. RIPEMD with key sizes 128 bits and 160 have been developed to replace the 128 bit MD algorithms. Both RIPEMD algorithms seem to achieve comparable performance to that of SHA-256, though clearly have shorter key sizes and so potentially less secure.

3.4 Summary

The results presented suggest that RSA-1024 and SHA-256 are the most suitable cryptographic algorithms for use during transactions in systems with real-time constraints. Almost any of the symmetric algorithms could be selected, but IDEA was shown to be the fastest in our evaluation.

4. Hybrid system

Web Services are built on open standards to provide a generic way of communication between heterogeneous environments. It therefore shows particular potential to be exploited within the e-commerce domain. In this section we investigate this further. In particular, we consider the combination of

cryptographic algorithms used in VeriSign's Trusted Services Integration Kit (TSIK) and evaluate them based on the level of security they provide as well as their performance and so conclude whether it is a suitable alternative for transactions with real-time constraints. TSIK's performance is evaluated through direct comparison with Java's Cryptography Extensions (JCE).

We therefore first detail the concepts that constitute such a hybrid system in Section 4.1. Section 4.2 analyses one element of the hybrid system, namely asymmetric cryptography, to aid in evaluating the level of security provided by TSIK, as detailed in Section 4.3, as well as understanding the results in Section 4.4. Final conclusions are drawn in Section 4.5.

4.1 System functionality

The hybrid system exhibits the following functionality, in which the techniques detailed in Section 2 are combined to achieve a more effective security solution through signing, verifying, encryption and decryption. They are combined as follows:

The key, in symmetric cryptography, can be securely transported using public key cryptography by encrypting the symmetric key using the receiver's public key. The receiver, and only the receiver, can then first decrypt the symmetric key using his private key and then decrypt the message using the decrypted symmetric key. Also note that only the key, which is relatively short, is encrypted using public key cryptography and so reduces encryption overhead.

The message digest, produced by the hash function, can be encrypted using an asymmetric cryptography algorithm to avoid and interception attack. Thus, if the message digest is encrypted using the sender's private key, only the message can be replaced during transit and not the message digest, since the interceptor does not have the sender's private key to encrypt the new message digest.

Generating a message digest and then encrypting the message digest using a private key is referred to as signing the message.

Decrypting the message digest using the sender's public key, generating a new message digest of the received message and then comparing the digests is called verifying the message. The performance results of these two techniques, among others, are analysed in this paper.

Sender authentication is achieved when the sender's public key is signed by a mutually trusted third party. The receiver can then verify the public key as the third party's public key is trusted.

4.2 RSA [Rivest1978]

Understanding the security implications and performance results in Section 4.3 and Section 4.4 requires a deeper understanding of public key cryptography. In particular RSA, which was developed by Ron Rivest, Adi Shamir and Leonard Adleman in 1977 and is used by VeriSign's TSIK toolkit. We do not explain all the details of RSA, but instead focus on the particular use of RSA in our measurement setup.

4.2.1 The algorithm [Rivest2003, Sun2005]

- Choose 2 large primes p and q such that $pq = N$
- Select 2 integers e and d such that $ed = 1 \text{ mod } \phi(N)$
 - Where $\phi(N) = (p-1)(q-1)$ is the Euler totient function of N

In general, N is called the modulus, e the public exponent and d the private exponent. The public key is the pair (N, e) which is made public and the private key is the pair (N, d) which is kept secret.

RSA encryption and decryption explained in context of the experiment scenario (Section 4.1):

Encryption:

The symmetric key M :
Encrypted key = $M^e \text{ mod } n$

The message digest M :
 Encrypted digest = $M^d \bmod n$

Decrypting:

The symmetric key C :
 Decrypted key = $C^d \bmod n$

The message digest C :
 Decrypted digest = $C^e \bmod n$

Where M is the key or digest converted to an integer according to [RSALab2002], C the encrypted key or digest and n the particular modulus, chosen to be either 512, 1024, 2048, 3072 or 4096.

In particular, it should be noted that encrypting the key and encrypting the message digest is not the same function as one uses the public- and the other the private exponent. Therefore, encrypting the symmetric key and decrypting the message digest (in the verification process) is mathematically equivalent as they both use the public exponent. The same can be said for encrypting the message digest (in the signing process) and decrypting the symmetric key as they both use the private exponent.

RSA operation time greatly depends on the length of e and d [Freeman1999], such that longer exponents incur much larger time overheads. It would therefore be desirable to use smaller values for e and/or d if possible.

4.2.2 Smaller public exponent

We consider how the length of the public exponent affects security as both security mechanisms (Section 4.3) exploit this to achieve faster symmetric key encryption and message verification. The smallest possible value for e is 3 [Boneh1999]. This can however weaken RSA confidentiality assertions. In particular, if $M < \sqrt[e]{N}$ the plaintext can easily be recovered [Rivest2003]. Hastad's broadcast attack can be mounted if k cipher texts, encrypted with the same public exponent, can be collected such that $k \geq e$. [Boneh1999]. The Chinese Remainder Theorem (CRT) can then be used to recover the plaintext message [Eastlake2001, Boneh1999]. A defence against such attacks

would be to 'pad' the message using some random bits [Bellare1994]. Coppersmith imposed further restrictions on this in his "Short Pad Attack" which concludes that for $e = 3$ an attack can still be mounted, even though a random set of bits are used, if the pad length is less than $1/9^{\text{th}}$ of the message length [Boneh1999]. PKCS#1 [Jonsson2003, RSALab2002] does however propose the use of Optimal Asymmetric Encryption Padding (OAEP) [Bellare1994] for new applications and PKCS1-v1_5 for backward compatibility with existing applications.

Although $e = 3$ can provide adequate security, if necessary precautions are taken, the current recommendation is $e = 2^{16} + 1$ [Boneh1999] which is still small, requiring only 17 multiplications, but big enough to solve the above problems at the cost of a slight increase in encryption time. Short public exponents are not however a concern for signature schemes [Eastlake2001, Rivest2003].

4.2.3 Smaller private exponent

A shorter private exponent would result in faster key decryption and message signing. Typically the private exponent is the same length as the modulus regardless of the public exponent length. Wiener [Wiener1990] has however shown that if $d < \frac{1}{3}N^{0.24}$ the private exponent can be obtained from the public key (N, e). Since N is typically 1024 bits long, d must be at least 256 bits long. More recently, Boneh and Durfree have shown this to be closer to $d < N^{0.292}$ [Boneh2000, Sun2005] and predicted the likely final result to be closer to $d < N^{0.5}$ [Boneh1999, Boneh2000].

Other techniques used to decrease algorithm operation time include the use of the Chinese Remainder Theorem [Boneh1999], know as RSA-CRT, which is said to be approximately 4 times faster than using standard RSA algorithms [Sun2005]. Rebalanced RSA-CRT can also be used and tries to shift the cost towards the usage of the public exponent e [Boneh2002, Wiener1990].

4.3 Security software analysis

Java keytool, Java's Key and Certificate Management Tool, is used to create the Java keystore, with appropriate key pairs, used by TSIK and JCE. The keytool generates key pairs where N is user specified (512, 1024 or 2048), d is the same length as N and e defaults to $2^{16} + 1$ (i.e. 17 bits long). As stated in Section 4.2.2 and 4.2.3, these values are adequate and it is currently recommended that the user selects the modulus to be at least 1024 bits.

TSIK 1.10 provides additional functionality, above that of the Java Cryptography Extensions (JCE), to construct valid XML messages after encryption/decryption or signing/verifying. These messages conform to the W3C XML Signature and Encryption specifications [W3C2002]. TSIK supports Triple DES (in cipher block chaining mode) for symmetric encryption, as defined by W3C [W3C2002]. Using a key length of at least 112 bits will currently provide sufficient security. Triple DES is however relatively slow compared to other more recent contenders such as AES [Aslam2004]. Conversely, it has stood the test of time and so is potentially a more reliable solution.

Only SHA-1 is provided for message digest generation (digest length of 160 bits). SHA-1 has very recently been shown to be less secure than predicted and it is recommended that SHA-256 or above should be used [Lenstra2005]. RSAES-PKCS1-v1_5 algorithm, specified by W3C [W3C2002] and [Kaliski1998], is used as the RSA standard. As stated in Section 4.2.2 above; if backward compatibility is not an issue OAEP should be used in preference to PKCS1-v1_5. However, PKCS1-v1_5 provides adequate security assuming the programmer is aware of certain issues. Also, [Kaliski1998] indicates that RSA-CRT is used.

JCE does not support the creation of valid XML messages but supports various symmetric key algorithms including AES, Triple DES and RC5. It also supports SHA-1, SHA-256, SHA-512 and MD5, amongst others, for message digest generation. It also

specifies that the padding is applied according to [RSALab2002]. RSA-CRT is also used.

4.4 Performance analysis

The following section details a comparative evaluation of the performance of VeriSign's TSIK toolkit with respect to the standard Java Cryptography Extensions (JCE) in order to identify whether TSIK is a viable tool to secure time-constrained online transactions.

4.4.1 Environment

All experiments were run on a 3GHz Intel Pentium 4 with 1GB RAM, running Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2-b28) on top of Linux Fedora Core 2. We used *The Legion of the Bouncy Castle* [Legion2005] as the Java RSA provider for both JCE and TSIK, and used Apache Axis 1.2 to generate the appropriate WSDL interface for the web service, which was hosted on Tomcat 5.

Axis was used to both generate the appropriate SOAP messages, from the Java code and TSIK XML documents, to be sent to the web service, also known as the server, and to generate the SOAP messages which are sent back from the web service to the client. We took performance measurements on the client and server side where the TSIK and JCE implementations reside. Message transmission and conversion delays were not measured.

4.4.2 Experiments

We set up three experiments, as detailed below.

Experiment 1:

In experiment 1 we analyse the performance of Triple DES, as function of message size:

- Client side: Message plaintext encrypted using Triple DES with a keysize of 168. Symmetric key encrypted using an RSA public key (Modulus 1024)
- Server side: Encrypted symmetric key decrypted using RSA private key (bit length 1024) and cipher text then decrypted.

Experiment 2:

In experiment 2 we analyse the combined performance of SHA-1 and RSA algorithms, as a function of the message size:

- Client side: Message signed using SHA-1 and RSA private key (bit length 1024)
- Server side: Message verified using SHA-1 and RSA public key

Experiment 3:

In experiment 3 we analyse how the modulus size affects the performance of RSA during signature creation and verification:

- Client side: Message signed (as in experiment 2) using RSA key sizes 512, 1024 and 2048.
- Server side: Message verified.

4.4.3 Results

We executed the above experiments for TSIK as well as JCE. We repeated the first two experiments for messages with a range of plaintext sizes, namely 2, 4, 8, 16, ... , 512 and 1024 kB. Experiment 3 was done using a 2 kB plaintext size. The results are shown in the graphs below. It should be noted that all points in Figures 6 and 8 exhibit confidence intervals of 3 milliseconds and points in Figures 7 and 9 exhibit confidence intervals of 0.1 milliseconds. Both with probability 0.9 (where 1.0 is certain).

Experiment 1:

Figure 6 shows that JCE performs noticeably better for large file sizes. It also shows that Triple DES encryption takes longer than decryption in both cases (TSIK and JCE). Note that the graph also indicates that for very large messages it is decryption that takes longer when using TSIK. We have no precise explanation for this, but suspect it has to do with the particulars of the implementation.

For RSA we see the opposite effect. Figure 7 indicates that RSA encryption takes less time than decryption. As we hinted at earlier in this paper, that is caused by the size of the keys used in encryption and decryption. For encryption, the public key is used, which has a

small public exponent of 17 bits. When comparing TSIK with JCE, we see that the differences are minimal. Decryption varies by an average of about 1 millisecond between the implementations and encryption even less.

Experiment 2:

Figure 8 shows that signing takes more time in both cases. This is once again expected as the messages are signed using the large 1024 bit RSA private key. Encrypting the message digest should take constant time for each file size and so the graph pattern should be wholly due to SHA-1 hashing. Whereas signing and verification time increase steadily for JCE, TSIK performs markedly worse for large file sizes.

Experiment 3:

Figure 9 shows that doubling the RSA key size causes signing time to increase rapidly whilst having little effect on the verification time. This can partly be explained by the fact that doubling the key size effectively doubles the length of the private exponent (used in signing) whilst keeping the public exponent length constant.

4.5 Summary

TSIK is a toolkit to aid secure Web Service interactions. We have shown, through performance measurements, that TSIK has comparable performance to Java's Cryptography Extensions (JCE). Its performance is similar to JCE, except that it slows down when processing messages with large plaintext sizes. It also provides adequate confidentiality, non-repudiation and sender authentication guarantees through the use of Triple DES and RSA, though should consider using SHA-256 for message verification in future releases as is suggested in recent literature [Lenstra2005]. With respect to technical ability, TSIK appears to be a viable and competitive option in securing web based business interactions.

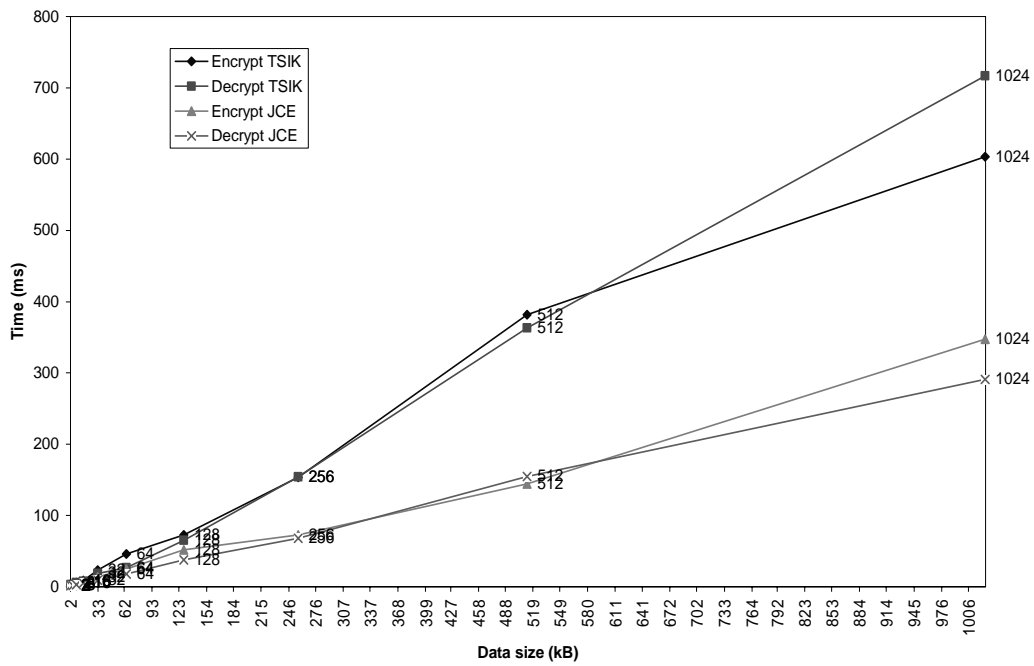


Figure 6: Triple DES encryption time

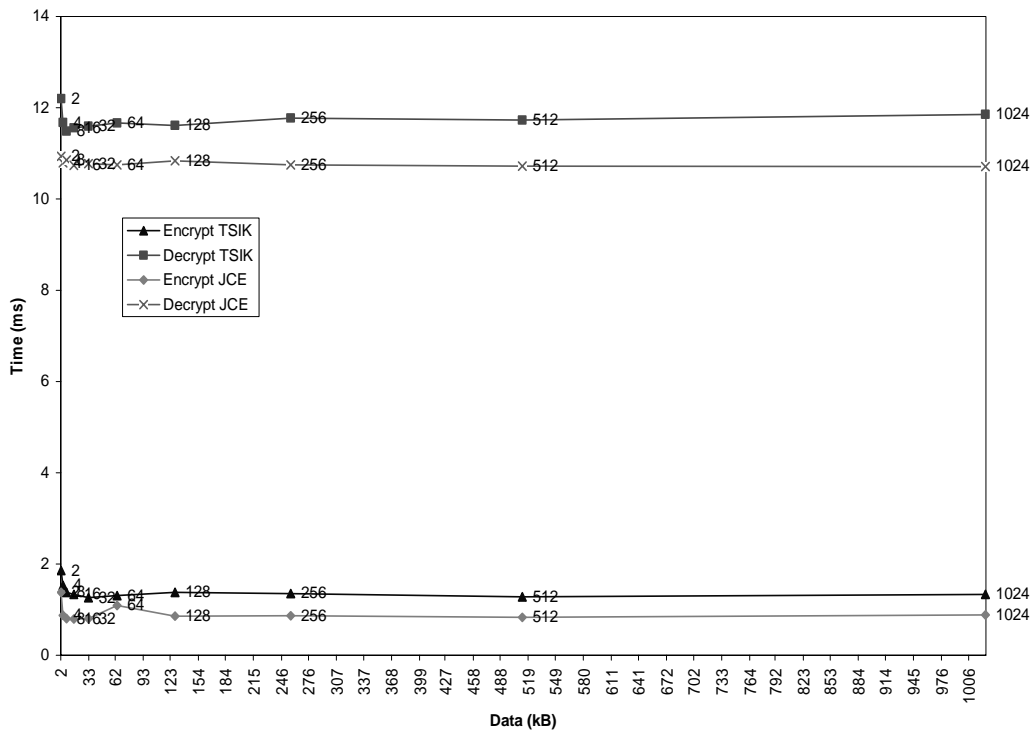


Figure 7: RSA-1024 encryption time of 168 bit Triple DES key

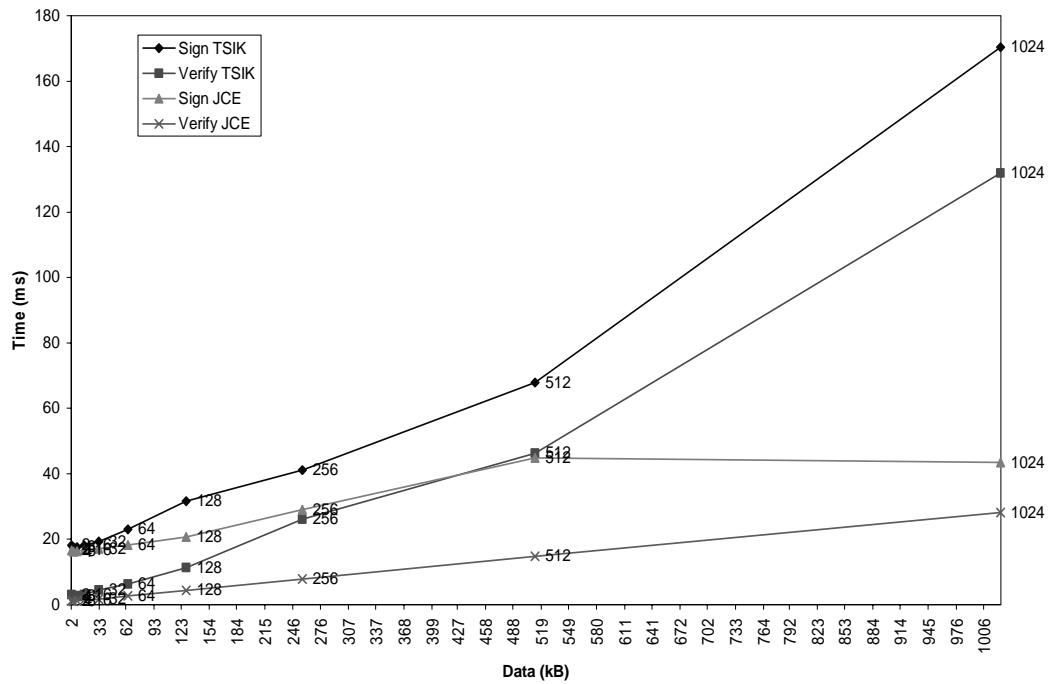


Figure 8: Message signing/verifying (using SHA-1 and RSA-1024)

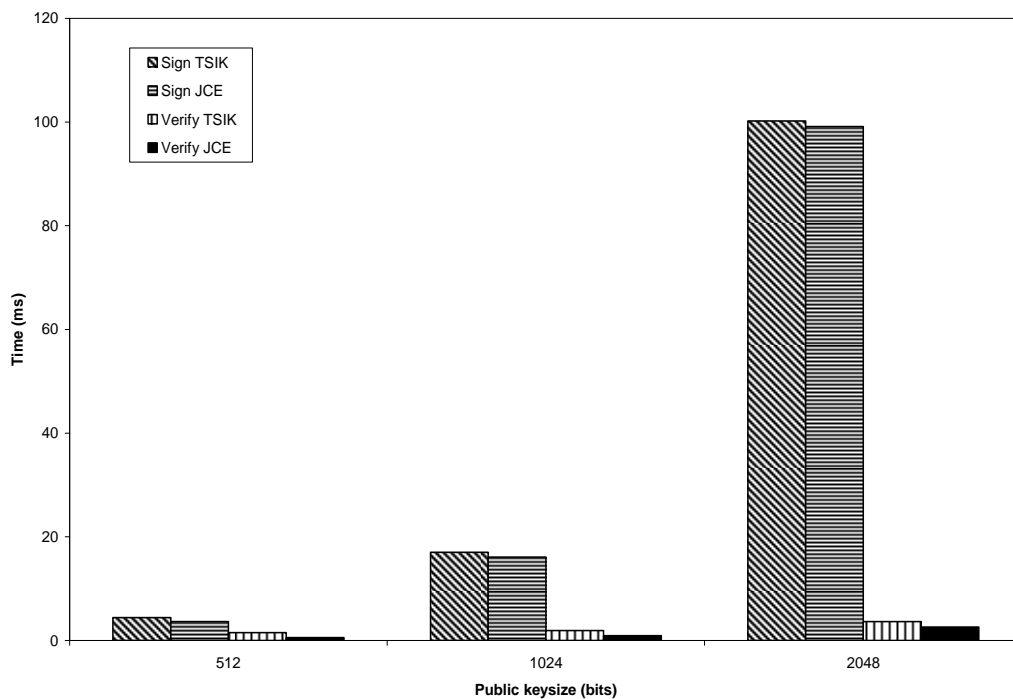


Figure 9: Message signing/verifying (2kB message size)

5. Conclusion

In comparing the performance of encryption algorithms we observed that IDEA was the fastest algorithm available in the distributions we tested. In fact most symmetric algorithms achieved better performance than Triple DES, which was used in our experiments with TSIG and JCE. Furthermore we found that the particular implementation of the algorithm also had a significant impact on the execution time. It therefore seems likely that further improvements can be made to TSIG to make it more suitable for real-time online transactions by selecting algorithms such as IDEA and SHA-256 and also providing fast implementations of those algorithms.

Bibliography

- [Adams2003] C. Adams and S. Lloyd, *Understanding PKI: Concepts, Standards, and Deployment Considerations*, second edition, Addison-Wesley, 2003.
- [Aslam2004] J. Aslam, S. Rafique and S. Tauseef-ur-Rehman, Analysis of Real-time Transport Protocol Security, *Information Technology Journal* 3 (3):311-314, 2004.
- [Bellare1994] M. Bellare and P. Rogaway, Optimal Asymmetric Encryption, In EUROCRYPT '94, *Lecture Notes in Computer Science* volume 950, pages 92-111, Springer-Verlag, 1994.
- [Bloom2002] J. Bloomberg, Testing Web Services Today and Tomorrow, *Rational Edge*, October 2002.
- [Boneh1999] D. Boneh, Twenty Years of Attacks on the RSA Cryptosystem, In *Notices of the American Mathematical Society* (AMS), Vol. 46, No. 2, pp. 203-213, 1999.
- [Boneh2000] D. Boneh and G. Durfee, Cryptanalysis of RSA with Private Key d Less than $N^{0.292}$, *IEEE Transactions on Information Theory*, 46(4):1339-1349, July 2000.
- [Boneh2002] D. Boneh and H. Shacham, Fast Variants of RSA, *CryptoBytes*, 2002, Vol. 5, No. 1, Springer, 2002.
- [Cryptix2005] Cryptix JCE, 2005.
<http://www.cryptix.org/>
- [Eastlake2001] D. Eastlake, RSA/SHA-1 SIGs and RSA KEYS in the Domain Name System (DNS), *IETF Network Working Group*, RFC 3110, May 2001.
- [Freeman1999] W. Freeman and E. Miller, An Experimental Analysis of Cryptographic Overhead in Performance-critical Systems, *MASCOTS*, October 1999.
- [Jonsson2003] J. Jonsson and B. Kaliski, Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography, Specifications Version 2.1, *IETF Network Working Group*, RFC 3447, February 2003.
- [Kaliski1998] B. Kaliski and J. Staddon, PKCS #1: RSA Cryptography Specifications Version 2.0, *IETF Network Working Group*, RFC 2437, October 1998.
- [Legion2005] The Legion of the Bouncy Castle, release 1.30, 2005.
<http://www.bouncycastle.org>
- [Lenstra2005] A. Lenstra, *Further Progress in Hashing Cryptanalysis*, February 26, 2005.
- [Rivest1978] R. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-key Cryptosystems, *Communications of the ACM*, 21(2):120-126, 1978.
- [Rivest2003] R. Rivest and B. Kaliski, *RSA problem*, To appear in *Encyclopedia of Cryptography and Security* (Kluwer).
- [RSALab2002] RSA Laboratories, *PKCS#1 v2.1: RSA Cryptography Standard*, June 14, 2002.
- [Sun2005] H.-M. Sun and M.-E. Wu, An Approach towards Rebalanced RSA-CRT with Short Public Exponent, *Cryptology ePrint Archive*, 053/2005.
<http://eprint.iacr.org/>
- [SunJCE2005] Sun Microsystems, Inc, Java™ Cryptography Extension, 2005.
<http://java.sun.com/products/jce/index.jsp>
- [Wiener1990] M. Wiener, Cryptanalysis of Short RSA Secret Exponents, *IEEE Transactions on Information Theory*, 36:553-558, May 1990.
- [W3C2002] W3C Recommendation, *XML Encryption Syntax and Processing*, <http://www.w3.org/TR/xmlenc-core>, 10 December 2002.
- [Schneier1996] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, Wiley, 1996.