A Parallel Signcryption Standard using RSA with PSEP

Anshuman Rawat and Shabsi Walfish

May 11, 2003

Abstract

We present a proposed standard for signcryption (a joint signature and encryption primitive) using the Probabilistic Signature and Encryption Padding (PSEP) scheme. The standard supports signcrypting short and long messages, associated data (“labels”), and key reuse (a single RSA key suffices for both signcryption and designcryption). The proposed scheme is provably secure under the strongest known definitions of security for signcryption. Furthermore, the standard is backwards compatible with the existing PKCS#1 public key infrastructure, and provides support for senders and receivers with different RSA modulus lengths. We motivate the general use of the signcryption primitive with practical issues, and demonstrate an application to key exchange protocols such as that carried out by SSL. In particular, we suggest a key exchange protocol designed to minimize the round complexity and computational cost. We conclude with a brief analysis of various techniques that can be used to increase the efficiency of our signcryption primitive.

Introduction

Until recently, cryptographic operations for signature (guaranteeing authenticity of messages) and encryption (guaranteeing the privacy of messages) were considered only as separate building blocks (“primitives”). Applications requiring both authenticity and privacy would generally compose these blocks directly to achieve those security requirements. In practice, these requirements often go together. This observation motivated the study of a new cryptographic primitive known as *signcryption*, designed to achieve simultaneous privacy and authenticity in the public key cryptographic setting. Signcryption was first introduced by Zheng [1] in 1997, with the expressed goal of achieving a signcryption scheme with a smaller “cost” than a sequential composition of signature and encryption. The signcryption primitive itself is also a useful abstraction for higher-level protocol design. To date, no complete standard in the style of the PKCS#1 public key encryption and signature standard [2], has been proposed for signcryption. This paper proposes such a signcryption standard that is fully backwards compatible with existing PKCS#1 RSA public key infrastructures, and motivates the new standard with real world applications to efficient key exchange protocols (such as that used by SSL).

This scheme we propose is based on the work of Dodis, Freedman, and Walfish [3] who present a paradigm known as padding-based parallel signcryption. While the primary computational advantage of padding-based parallel signcryption over straightforward composition of encryption and signature lies in its parallelization of the signature and encryption operations (which is rarely of practical value), the advantages of the scheme in terms of security, flexibility, and compatibility are sufficient to merit its use. For a detailed comparison of the advantage of padding-based parallel signcryption in relation to other signcryption schemes, see [3]. Most importantly, our proposed standard has been proven to be semantically secure against adaptive chosen ciphertext attacks and to provide strong unforgeability against chosen message attacks. Both of these security guarantees are provided in the full multi-user, insider-security setting (described in further detail below). Our proposed signcryption scheme also includes support for associated data (a “label” that is inextricably bound to the ciphertext), as well as support for long messages (such as emails).

Finally, we evaluate the performance of our proposed signcryption scheme in the context of a simple key exchange protocol designed to provide an efficient SSL session key exchange protocol replacement. Our implementation is based on the OpenSSL library, modified to include support for multi-prime RSA operations for benchmarking purposes. We show that substantial performance improvements can be realized for server-side computations required during the exchange. Additionally, at least one round of communication is eliminated. These improvements are of great practical significance since the major bottleneck for many web servers is the excessive computational overhead of performing an SSL key exchange.

Security of Signcryption

For the purposes of this paper, we will consider a signcryption scheme to be secure when it is provably both IND-CCA2 and sUF-CMA secure against insiders in the multi-user setting. Security against insiders implies that the authenticity of the scheme is protected even if the recipient of the ciphertexts is in fact malicious, and that the privacy of the scheme is protected even if the sender of the ciphertexts is malicious. Clearly, this notion is a stronger notion of security than outsider security (which does not provide these guarantees). Multi-user security implies that no "identity fraud" can occur in a setting where multiple users may have keys (which is generally the setting of interest). In this setting, IND-CCA2 security implies that any polynomial time adversary has at most negligible success advantage in the following game: The adversary is first allowed to query a user R to signcrypt arbitrary messages to arbitrary recipients, or to designcrypt arbitrary ciphertexts of the adversary's choosing. The adversary then selects two messages, M_0 and M_1, and asks the challenger to signcrypt one of them at random (with R as the recipient, using a sending secret key of the adversary's choice) and to return the corresponding "challenge" ciphertext to him. The adversary may then continue to query R for arbitrary signcryptions and designcryptions, subject to the restriction that he may not request the designcryption of the challenge ciphertext. If no polynomial time adversary can guess whether the challenge ciphertext corresponds to M_0 or M_1 with better than negligible advantage over random guessing, we say that the signcryption scheme is IND-CCA2 secure. The sUF-CMA security definition implies that any polynomial time adversary has at most negligible success advantage in the following game: The adversary is allowed to query user S to signcrypt messages of his choice to recipients of his choice, or designcrypt arbitrary ciphertexts of his choice. Eventually, the adversary attempts to output a valid ciphertext that appears to be sent by S, but was not previously returned to the adversary by one of his queries to S.

Signcryption with associated data adds the additional guarantee that information contained in a label L is considered to be part of the ciphertext with respect to forgeries, but L may be public, and in fact, L may not even be sent along with the ciphertext. A recipient must check the validity of a signcryptext using the correct label. Our proposed signcryption scheme satisfies the property that it is not even feasible for the recipient to recover the signcrypted message M without first providing the correct label for the designcryption operation.

A signcryption scheme satisfying the above security requirements represents an extremely resilient cryptographic primitive that may be safely used in a wide variety of contexts. As per the analysis in [4], a simple composition of signature and encryption will not satisfy these requirements. In particular, any composition of PKCS#1 encryption and signature schemes will not satisfy these definitions of security (even using the commit then encrypt and sign paradigm proposed in [4]). Thus, to provide for the greatest possible utility of the signcryption primitive, we propose to use the construction provided in [3], which provides all these security guarantees.

The Proposed Standard: RSASCS-PSEP and RSASCS-LPSEP

We propose an amendment to the existing PKCS#1 v2.1 standard [2] which adds support for a new signcryption primitive, which uses one of two new encodings: RSASCS-PSEP for short messages and RSASCS-LPSEP for long messages. These schemes are based on the Probabilistic Signature Encryption Padding (PSEP) scheme first presented in [3]. A low-level description of these encoding schemes will be provided in a future standards document. As a high-level overview, using the notation established in the PKCS#1 document, an explanation of the RSASCS-PSEP encoding of a message *M* to be signcrypted from a sender *S* to a recipient *R* using label *L* is given below. The notation || indicates string concatenation, and ⊕ represents a bitwise exclusive-OR operation.

Step 1: The short message M is split into two (zero-padded) parts, *M1* and *M2*
Step 2: A random bit string *seed* of *HashLength* octets in length is generated
Step 3: A data block *DB* is constructed as follows:
DB = *M2* || *seed*
Step 4: An encoded message block *EM1* is constructed as follows:
EM1 = Hash(*DB2*) || MGF(*seed*) ⊕ *M1*
Step 5: A label block LB is constructed as follows:
LB = *L* || *HashLength* || sender's public key || recipient's public key
Step 6: An encoded message block *EM2* is constructed as follows:
EM2 = MGF(LB || DB1) ⊕ *DB*

The function Hash( ) is any fixed length output CRHF that can be modeled as a random oracle that is defined in PKCS#1 standard (for example, the SHA1 hash function), with an output length of *HashLength* octects. The function MGF() is a "mask generating function" that extends a fixed length output collision resistant hash function (CRHF) to a variable length output CRHF. The details of the construction of the MGF( ) using Hash( ) are provided in the PKCS#1 v2.1 document [2]. The resulting blocks *EM1* and *EM2* are then encrypted with the RSA public key of the recipient and signed with the RSA private key of the sender (respectively). Thus, the final ciphertext is $C = \mathrm{RSA}_R(EM1) \,||\, \mathrm{RSA}^{-1}_S (EM2)$, where $\mathrm{RSA}_R(\ )$ denotes RSA encryption using the public key of the recipient and $\mathrm{RSA}^{-1}_S(\ )$ denotes RSA decryption using the sender's private key. Figures 1 and 2 below illustrate the construction of *EM1* and *EM2*.

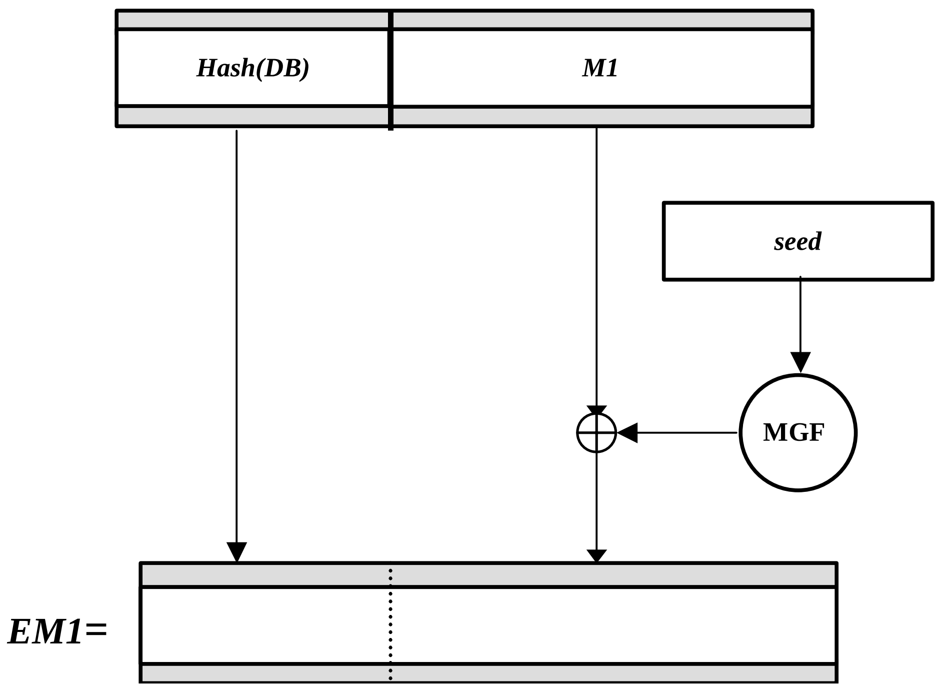

Figure 1: Constructing *EM1*.

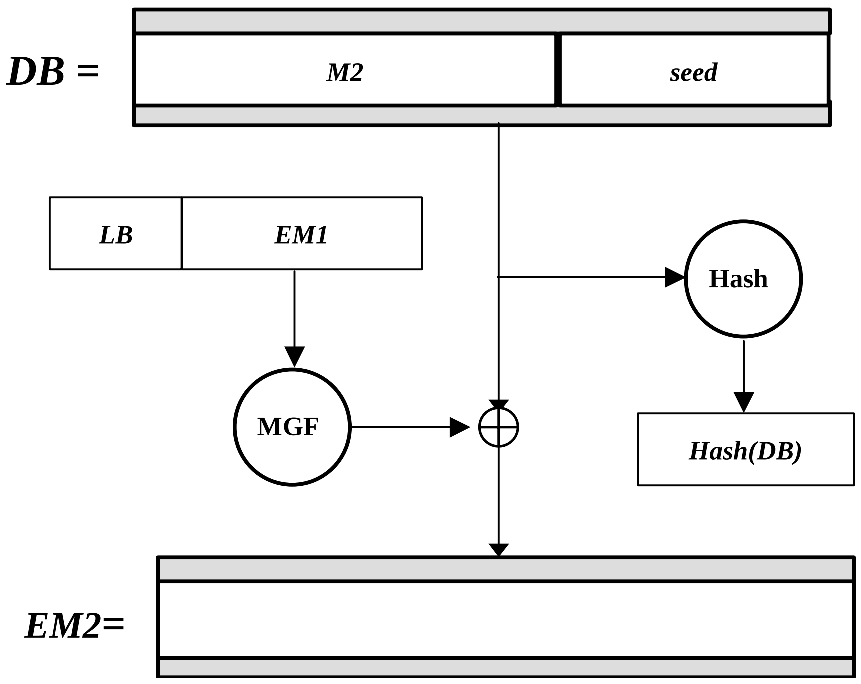

Figure 2: Construction of *EM2*

The designcryption operation essentially runs the construction in reverse. One important point of the scheme not clearly captured in the pictures above is that the full RSA public keys of both sender and recipient are bound to the signcryption of *M*, along with the label *L*, in the label block *LB*. This prevents “identity fraud” attacks, and provides increased security for users who may publish multiple public keys with the same RSA modulus since the full public key, including the public exponent, is included in *LB* (this is relevant for the Batch-RSA based schemes mentioned later).

Note that all lengths can be chosen “appropriately” to support mismatched sender and recipient RSA modulus lengths. This is important for legal reasons. For example, a sender might publish a certified public key using 1024-bit RSA, which it is allowed to use in some countries for the purposes of signing, whereas encryption operations are limited to 512-bit moduli by law. In this circumstance, if the sender wishes to signcrypt a message to a recipient with a 512-bit key, it need not generate and sign a temporary 512-bit key; instead, the sender can simply apply the signcryption primitive directly using it’s 1024-bit sender’s key and the 512-bit recipient’s key.

If the desired application of signcryption requires “non-repudiation”, RSASCS-PSEP can be used to achieve this (barring disclosure of the sender’s private key). A recipient who wishes to display the signature of the sender can simply decrypt *EM1*, and provide it along with his public key and the sender’s signature of *EM2* as evidence, which can then be verified. This does not require the recipient to reveal his private key in order to prevent the sender from repudiating his message.

The RSASCS-LPSEP encoding is virtually identical to the RSASCS-PSEP encoding. Rather than encoding the message *M* directly, it encodes a short random key that is used to encrypt the *M*. The encrypted copy of *M* is appended to the label *L* during the encoding process, so that the encrypted message is bound to the signcryption of the key required to decrypt it. The resulting scheme efficiently encodes long messages, and still provides the full security and flexibility benefits of RSASCS-PSEP.

Using RSASCS-PSEP for SSL Key Exchange

The SSLv3 (Secure Sockets Layer version 3) protocol is a commonly used protocol for establishing secure connections between client web browsers and web servers. In the common situation, web servers have certified RSA public keys representing their identities, whereas clients do not (and thus are usually not authenticated). A minimal SSLv3 handshake and key exchange protocol will proceed roughly as follows:

Round 1 (Client -> Server) :	Handshake messages
Round 2 (Server -> Client) :	Handshake + Server's Certified Public Key S
Round 3 (Client -> Server) :	RSA_S(fresh secret)

The server can use a signed temporary RSA public key instead of S is to ensure forward security (that is, even if the server's secret key is compromised at some point, all previous communications remain secure), but achieving full forward security would require the server to regenerate this key for every connection, which is computationally expensive. Another way of achieving full forward security is to employ a Diffie-Hellman Key Exchange, but this too is computationally expensive. Setting aside the issue of forward security, the computational requirements for the server mainly come from decrypting the fresh secret sent by the client in Round 2. Since RSA decryption operations are expensive (even for "low-exponent" RSA), the server is performing a very computationally expensive operation.

We suggest the following signcryption based protocol, which helps to minimize the computational expenses on the server side, in trade for more client-side computation, while retaining full forward security.

Round 1 (Client -> Server) :	Handshake messages + temporary client RSA key C
Round 2 (Server -> Client) :	Handshake + Server's Certified Public Key S + Signcryption(session key) from S to C

This protocol eliminates one round of communication by allowing the server to choose the session key. Furthermore, provided that the client uses fresh temporary RSA keys for every SSL connect (it takes approximately 1 second to generate a 1024-bit RSA key on a Pentium II running at 450Mhz), the protocol provides full forward security and precludes replay attacks. The server-side computational expenses are approximately the cost of a single RSA decryption (provided that the client uses low-exponent RSA, which renders the cost of an RSA encryption trivial compared to the cost of a decryption).

However, unlike the “best case” version of the SSLv3 protocol presented above, in our suggested protocol the server can easily employ the batch RSA technique described in [5]. This technique allows the server to “batch” signing operations together, performing one expensive computation to obtain the results for all of the operations. It requires the server to publish several separate certified public keys with different RSA public exponents but the same RSA modulus (one for each signing operation to be included in a batch). Our protocol can handle this transparently from the client perspective, since the server simply provides the appropriate public key to each client in the batch. The server can select any arbitrary set of pending signcryption operations to batch together. On the other hand, using batch RSA with the SSLv3 protocol is extremely inefficient since it would be difficult to properly schedule the different encryption exponents among the clients, and the server might have to wait a significant period of time for all the client responses to arrive for a given batch. Batch RSA is has been benchmarked at approximately 2.64 times the speed of standard RSA for 1024-bit keys [6]. Thus, our suggested protocol can easily be made more than twice as fast on the server side as the fastest version of the SSLv3 protocol.

Further Improvements

Further speed improvements to RSA decryption and signing operations, which apply to both the existing SSL key exchange protocol and our proposed protocol, include multi-prime and multi-power RSA, as well as rebalanced RSA. Multi-power RSA is not recommended due to security concerns [7]. In [6], we find that rebalanced RSA is approximately 3 times faster than standard RSA. We have benchmarked multi-prime RSA, which is widely accepted and has become part of the PKCS#1 v2.1 standard, at more than 1.7 times as fast as standard RSA. Significantly, all these techniques can be combined, and we expect the result to be more than 10 times as fast as standard RSA using multi-prime, rebalanced, and batch RSA, or approximately 4.4 times as fast as standard RSA using just batch RSA and multi-prime RSA (which provides extremely reasonable security) for 1024-bit keys. Since the cost of our proposed key exchange protocol is approximately the cost of a single RSA decryption, we can use benchmarks of RSA decryption time to determine the maximum number of key exchanges a server can perform per second.

We modified an open source library for SSL, known as OpenSSL, to support multi-prime RSA as specified in the PKCS#1 v2.1 document, and benchmarked the resulting RSA decryption / signing performance on an Intel Xeon 2 Ghz processor running Redhat Linux 7.3. The results are summarized in Table 1 below.

	1024-bit modulus	**2048-bit modulus**	**4096-bit modulus**
2 primes	155	26	4
3 primes	264	49	8
4 primes	386	76	12

Table 1: RSA decryption (or signature) operations per second

The table indicates that a typical web server running on modern hardware should be able to perform over 260 key exchanges per second using a 1024-bit RSA modulus with 3 primes and our proposed key exchange protocol. Furthermore, if batch RSA were implemented, we anticipate this number would jump to over 675 key exchanges per second. In combination with rebalanced RSA we could exceed 1000 key exchanges per second, but it is unclear how safe this approach is from a standpoint of security. Similarly, the protocol can be made more efficient by using smaller RSA moduli (512 bit moduli are currently in widespread use, even though such moduli have been broken by distributed computing efforts in the past), although this would seriously compromise the security of the protocol. We note that the same Xeon 2 Ghz processor used for these benchmarks could perform a maximum of approximately 150 SSL key exchanges per second in an unmodified 512-bit RSA OpenSSL configuration (with Diffie-Hellman Key Exchange enabled).

Conclusion

In conclusion, we feel that our suggested signcryption primitive has proven to be a useful one. Protocol designers can easily use it to achieve extremely powerful security guarantees in a wide variety of settings. It is efficient, backwards compatible, and easily implemented. Using the latest fast variants of RSA [6], the signcryption operation can be used to dramatically improve both the performance and security of RSA based key exchange protocols such as those in SSLv3. Furthermore, protocols using signcryption often have lower round complexity than their counterparts using plain signature and encryption operations (as evidenced by our removal of a round from the minimum round SSL key exchange). Protocols and applications that already rely on separate RSA signature and encryption operations can be made conceptually simpler, more efficient, and more secure in a straightforward fashion by simply replacing those operations with signcryption. Signcryption is inherently less likely to be abused by practitioners due to its extremely strong security guarantees, which more closely match the idealized view of the strength of cryptographic primitives most people find to be intuitive. Our scheme also provides many small hidden benefits that will likely find use in time. (For example, it is possible to send the portion of the RSASCS-PSEP encoding that needs to be signed over to another machine, using any authenticated channel, for it to be signed there. The machine performing the signing operation learns nothing about the ciphertext.)

While the signcryption standard we proposed is ideally suited to most RSA public key environments, a few drawbacks remain. For applications where extremely low bandwidth is available and short messages must be exchanged, the minimum ciphertext length of our scheme may be unacceptable. It may also be in appropriate for settings where it is impractical to perform the padding operation in the presence of the full RSA public keys of both the sender and recipient (i.e. extremely low memory environments such as smart cards). These drawbacks could have been further minimized at the cost of extra complexity in the implementation of the scheme. We believe these drawbacks are extremely minor in light of the many advantages of using RSASCS-PSEP, and thus we recommend the standard in its present form.

Bibliography

[1] Y. Zheng, Digital signcryption or how to achieve cost(signature & encryption) << cost(signature) + cost(encryption), in Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 165-179. Springer-Verlag, 17-21 August 1997.

[2] RSA Laboratories, PKCS #1 v2.1: RSA Encryption Standard, June 2002.

[3] Yevgeniy Dodis, Michael J. Freedman, and Shabsi Walfish, Parallel Signcryption with OAEP, PSS-R, and other Feistel Paddings, in *Cryptology ePrint Archive*, Report 2003/043, http://eprint.iacr.org/2003/043, March 2003.

[4] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars, Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, Lecture Notes in Computer Science. Springer-Verlag, 28 April – 2 May 2002.

[5] A. Fiat. Batch RSA. In G. Brassard, ed., *Proceedings of Crypto '89*, vol. 435 of LNCS pp. 175-185, Springer-Verlag, 1989

[6] Dan Boneh, Hovav Shacham. Fast Variants of RSA. RSA CryptoBytes, vol. 5, no. 1 (2002), pages 1–9.

[7] Personal communication from Victor Shoup, regarding the difficulty (or lack thereof) of factoring integers of the special form $N = p^2q$ used by multi-power RSA