Optimizing Multitenant Managed File Transfer Part One: Concepts and Approaches

David J. Walling
david.walling@yahoo.com

Abstract

This work presents software architecture concepts for optimizing cryptographic and related functions used in providing secure delivery of data in a managed, multitenant environment. A multitenant architecture for providing secure file transfer between commercial enterprises, regardless of the overarching application, presents several numerous security challenges as well as opportunities for increased efficiency compared to a multi-instance deployment model. The optimizations considered in this work include

- accelerating algorithms that reuse commonly computed values,
- increasing throughput by applying algorithms in parallel,
- more efficiently utilizing scarce resources by concurrently applying data transformations, and
- employing an adaptive performance tuning capability through real-time measurement and analysis.

Finally, this work recommends metrics for quantifying performance improvements gained by implementing the techniques described.

Table of Contents

1. Introduction. 3

1.1. Secure File Transfer. 3
1.2. Managed File Transfer using SaaS. 4
1.3. Multitenant Managed File Transfer. 6
1.4. Key Management Challenges. 7

2. Defining Architecture Objectives. 8

2.1. Analyzing Secure File Transfer 8
2.2. Managed File Transfer Differentiators. 9

3. A Multitenant MFT Architecture. 11

3.1. Security Algorithm Basics. 11
3.1.1. *Compressing Data.* 11
3.1.2. *Computing a Message Digest.* 12
3.1.3. *Creating and Verifying a Digital Signature.* 12
3.1.4. *Bulk Message Encryption and Decryption.* 12
3.1.5. *Encrypting and Decrypting a Symmetric Key* 13
3.2. The Secure Transform Cell. 13
3.3. Data Transform Streams. 15
3.4. The Multitenancy Resource Cube. 16
3.5. Algorithm Optimization Example. 17
3.5.1. *Selection Criteria.* 17
3.5.2. *Modular Exponentiation.* 17
3.5.3. *Modular Exponentiation Algorithm.* 18

4. Summary. 20

5. References. 21

Glossary of Abbreviations 22

Index 23

1. Introduction.

Information security is an increasingly critical aspect of modern business software. Since electronic commerce necessarily involves the transfer of sensitive information between enterprises, tremendous attention continues to be devoted to properly and efficiently securing information in motion as well as at rest.[1] Information security is also a very broad topic that encompasses many disciplines including disaster recovery, access control, cryptography, incident detection and reporting, to name only a few. The scope of this work is limited to software architecture principles designed to optimize security-related functions that provide the foundation of managed, secure file transfer.

1.1. SECURE FILE TRANSFER.

Secure File Transfer (SFT) can be reduced to four basic elements - privacy, authentication, integrity and non-repudiation of receipt. These basic elements are described in Table 1.

Element	Description
Privacy	Privacy is usually provided by encrypting that information in order to prevent unauthorized access to information.
Authentication	Authentication assures the recipient that information is authentic. That is, the identity of the originator is certain.
Integrity	Message integrity provides an assurance that the information has been received complete, intact and unaltered.
Non-Repudiation of Receipt	Non-repudiation of receipt occurs when the sender of information receives a provably authentic receipt from the recipient demonstrating the information was successfully received.

TABLE 1. Secure File Transfer Attributes.

Several distinct operations are performed to manifest the four elements of secure file transfer. The sections below describe optimizing these operations in a managed service setting. At the outset, it is important to note that these operations are inter-related and dependent on each other. Figure 1 illustrates this relationship.

[1] [GISS] p. 25.

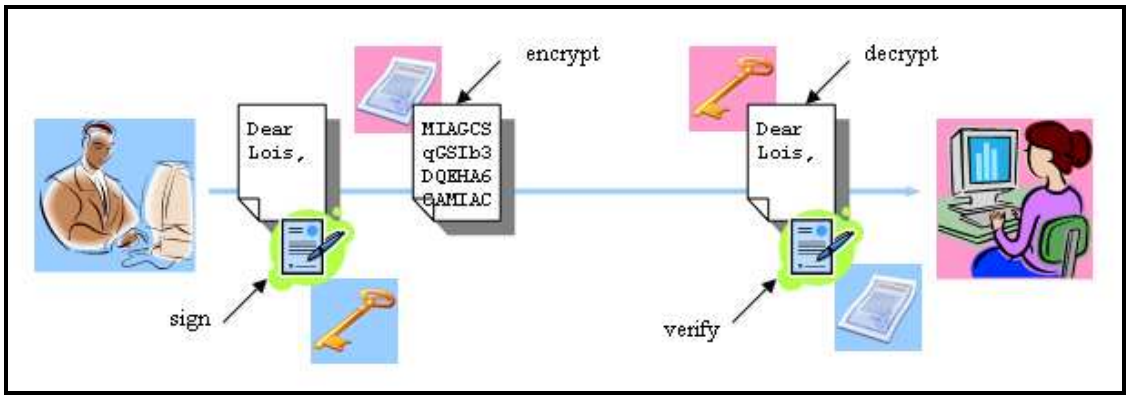

FIG. 1. Secure File Transfer Basics.

The process of conducting a secure file transfer follows these steps:

- A message digest is computed over the content of the document.
- The digest is encrypted using the sender's private key to form a digital signature.
- The document is encrypted with the recipient's public key found in the recipient's public certificate.
- Upon receipt, the message is decrypted.
- After decrypting message, the sender's signature can be verified by decrypting it with the public key found in the sender's certificate.
- Finally, the recipient can return a signed receipt containing the message-digest computed by the recipient. This receipt proves that the message was sent successfully.

1.2. MANAGED FILE TRANSFER USING SAAS.

Deploying SFT as a managed service, as software-as-a-service, or SaaS, can eliminate or reduce costs of installation, maintenance and support of software for both the software vendor and consumer, providing a "win-win" scenario. For the vendor, cost savings are maximized by reducing the complexity of systems and optimizing the use of shared computing resources.

Managed SFT, or MFT, provides its own unique architectural challenges. For example, when secure file transfer is implemented in-house, these solutions must scale one-dimensionally, in a hub-and-spoke model. However, when implemented as an MFT SaaS solution, software must scale in several dimensions due to multi-instance or multitenant architecture demands. Identifying these demands is essential to prioritize our architecture objectives. Here, we organize our design objectives into three areas, each of which relates to a key capacity metric. These metrics are (1) the number of connection points, or "end points", (2) transaction volume, and (3) file size.

The number of end points generally represents the number of distinct business entities subscribing to the managed file transfer service. This is a key capacity metric because it determines both (a) the probable frequency of certificate updates – an important administrative function related to *key management*, and (b) the number of public key-bearing certificates that will be maintained by the system. This number is important because it is a factor in determining storage requirements and whether security credential searching might become a performance bottleneck. The transaction volume is a key capacity metric because it is a factor in determining both (a) the number of network connections to be established, and (b) the number of unique symmetric keys needing to be generated to perform the bulk encryption of data, assuming the best-practice of using a unique, random symmetric key for each encryption. The average and maximum size of files being transferred is a key capacity metric because it directly influences both (a) the average duration of a transfer, which, in turn, will contribute to determining the average number of concurrent network connections that must be sustained and (b) the disk and memory utilization required. Table 2 summarizes these fundamental challenges of Managed File Transfer.

Capacity Metric	Challenges	Opportunities/Responses
Connection points	Increasing labor required to manage the interconnecting community.	Invest in labor-saving out-of-band transfers to auto-update expiring certificates.
	Increasing number of concurrent connections may exhaust network resources.	Look for optimizations that shorten the overall connection lifetime.
Transaction Volume	Increasing number of concurrent connections may exhaust network resources.	Explore protocol options allowing reuse of connections.
	Increasing number of unique, random bulk encryption key generation and asymmetric signing operations contribute to high CPU utilization.	Look for optimizations that eliminate redundant calculations. Preserve pre-computed values used repeatedly.
File Size	Large files increase time required to compute message digest and perform bulk encryption, leading increase in concurrent transfers.	Avoid processing large files in stages. Reduce disk and memory usage by processing large files as they are received from or written to the network.
	Intermittent network outage rate more likely to interfere with transfer of larger files.	Use open-standard methods for sending large data piecemeal, such as HTTP chunked-encoding. Send large files in parallel over separate connections to shorten total elapsed transfer time.

TABLE 2. Core Managed File Transfer Metrics.

1.3. MULTITENANT MANAGED FILE TRANSFER.

SaaS vendors provide services using both multi-instance and multitenant architectures. A multi-instance architecture provides maximum isolation for each customer's environment. A multitenant architecture maximizes reuse through shared resources. Multitenancy can refer to a software architecture that provides functionality to multiple distinct operating business entities that represent separate, and possibly competing, commercial interests. There are obvious security implications when implementing MFT as a multitenant architecture.

Security credentials maintained in a multitenant managed file transfer system include private keys used for (a) digital signature creation and (b) decrypting symmetric encryption keys and public keys, usually contained in digital certificates, used for (a) encrypting symmetric keys and (b) verifying digital signatures. Private key in a multitenant MFT architecture are associated with the "tenants" – those transacting entities thought of as being "housed" by the system. Public keys (certificates), however, would be associated both for tenants and for remote connection points, or "partners."

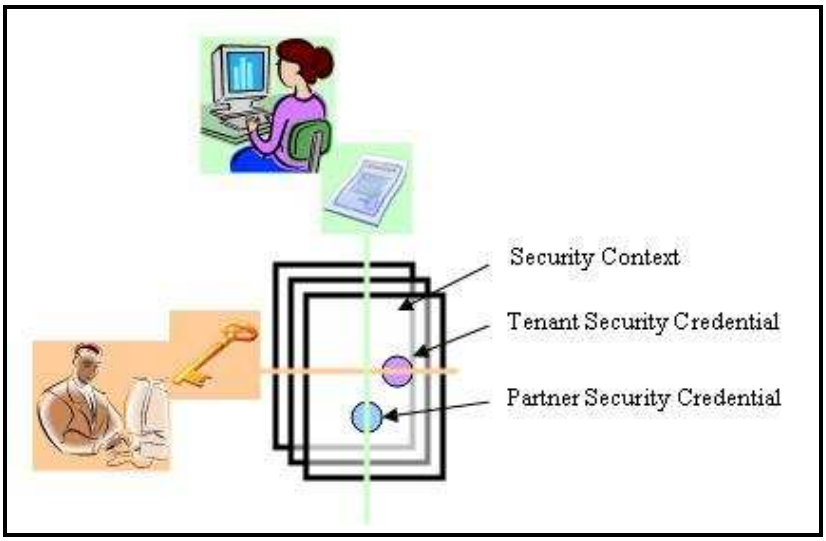

FIG. 2. An MFT Security Context.

Each tenant may securely exchange data with one or more partners. Any partner may securely exchange data with more than one tenant. Therefore, if we refer to the set of partners with which a tenant exchanges data as a tenant's "transfer domain", these domains may overlap or be completely isolated. Figure 2 above illustrates (a) these two dimensions with one tenant and one partner, and (b) how the combined use of a tenant's private key and a partner's public key create an intersection, or "security context." Figure 3 below illustrates the multiplication of security contexts in a multitenant architecture.

In Figure 3, tenants A and B exchange data with partner 1. Each security context within this transfer domain will share the partner's public key. Since the same public key will be used by both tenants to encrypt data for Partner 1, optimization potential exists at the algorithm level within the Partner Transfer Domain during encryption and signature verification. Similarly, Tenant A exchanges information securely with partners 1 and 3. Each of these security contexts share the private-key of Tenant A. Optimization potential exists, therefore, at the algorithm level within Tenant Transfer Domain A during signing and decryption operations. An isolated transfer domain defines the relationship between Tenant C and Partner 2. The security credentials used in this domain are not shared.

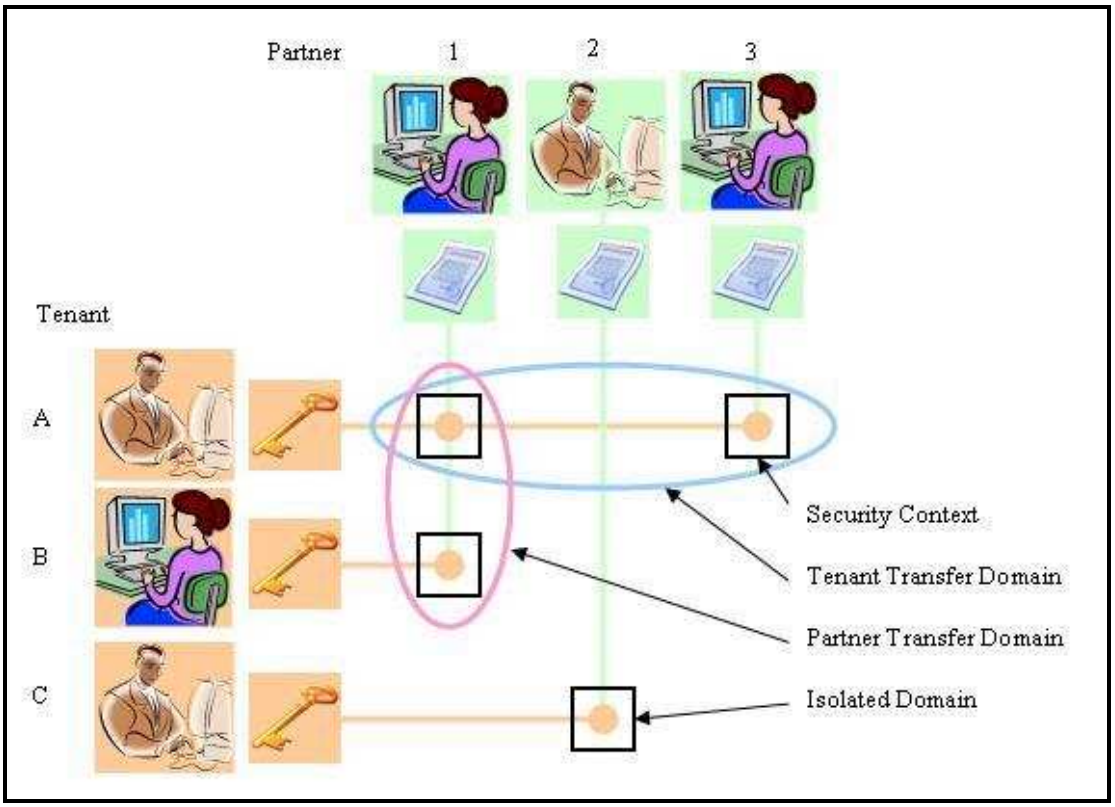

FIG. 3. Multitenant MFT Transfer Domains.

1.4. KEY MANAGEMENT CHALLENGES.

The most important aspect of this diagram is the relationship between the public and private components of the asymmetric key material used to accomplish each of the basic security operations, signing, encryption, decryption and signature verification. Maintaining an ongoing, valid association between private keys and public certificates at both ends of the secure file transfer is challenging. The difficulty of providing security measures that effectively protect private keys should not be underestimated. The frequency at which public certificates expire can lead to labor-intensive support

procedures. As we will explore further, multiplying these fundamental challenges in a high-volume managed, multitenant environment requires an architecture that can avoid excessive resource consumption and deliver consistently acceptable performance.

A customer's information security policy may impact a vendor's ability to offer services using a multitenant architecture if policies prohibit the comingling of either data or processes on shared resources. As this work will show, implementing multitenancy to satisfy the requirements of secure file transfer offers compelling processing efficiencies. These opportunities should not be viewed as competing with security policies. Instead, the opportunity to realize these gains should serve to incent architects to effectively address security policy concerns as well. This work draws attention to some specific areas where such opportunities and challenges exist.

2. Defining Architecture Objectives.

Having identified core attributes and metrics of Managed File Transfer, we now define architecture objectives based on (a) reviewing the nature of the fundamental operations involved in securing file transfer and (b) identifying potential capability differentiators. In other words, we can define what we "must" do to meet the essential requirements of SFT and what we "can" do to produce an architecture that is superior.

2.1. Analyzing Secure File Transfer.

To satisfy the requirements of Secure File Transfer, cryptographic and other related operations are applied to data. For privacy, encryption modifies the binary representation of information. To satisfy the other basic requirements, additional information is generated, such as message integrity check values, digital signatures, and signed receipts.

Encryption does not strip the data of meaning. It simply introduces a new factor, the decryption key, required to gain meaning from the data by restoring it to its original, meaningful, format. Since the meaning of the information is still present in the encrypted data, compromise must be guarded against by diligently protecting the decryption key. For this reason, key management is a critical part of information security. Good security policies will take proper key management into consideration.

For each of these operations, one or more standard algorithms are used. Table 3 illustrates some of the many possible algorithms that can be used for each of the basic functional requirements. Although an in-depth understanding of how these algorithms work is not required to appreciate the optimizations recommended in this work, it is important to note these algorithms and understand their significance.

Requirement	Operation	Algorithm(s)	Significance
Privacy	Compression	ZLIB	Although not required to encrypted data, compressing data prior to encryption enhances resistance to cryptanalysis and accelerates both bulk encryption and message digest computation.
	Content Encryption	AES, 3DES	Bulk data encryption typically uses a fast block cipher algorithm in which the same key is used both for encryption and decryption. Using a unique key for each encryption combined with a block chaining mode enhances resistance to cryptanalysis.
	Key Encryption	RSA	The symmetric content encryption key is sent along with the data and is protected using an asymmetric encryption method. In this kind of encryption, one half of a public/private key pair is used to encrypt the symmetric key. Using an asymmetric algorithm such as RSA, only the corresponding key in the pair can decrypt. Private keys must be secured and never shared. Public keys are usually distributed in the form of digital certificates that have limited terms and conditions of use.
Authentication	Digital Signature	SHA, RSA	A message digest is asymmetrically encrypted with the sender's private key so that every recipient who verifies this signature can be certain of the message's origin.
Integrity	Message Digest	SHA, MD5	At the heart of both a digital signature and a signed receipt is a message digest, computed over the original message using a hashing algorithm such that any alteration of the message can be detected by re-computing and comparing the value of the message digest.
Non-Repudiation of Receipt	Digital Signature	SHA, RSA	A receipt, digitally signed by the recipient, and carrying the recipient's computed message digest proves to the sender that the message was received intact.

TABLE 3. Basic Information Security Operations.

2.2. MANAGED FILE TRANSFER DIFFERENTIATORS.

Considering the challenges and opportunities identified above, we can now summarize our key architecture objectives. To organize these, objectives are categorized within several measurable criteria. Table 4 specifies our architecture objectives organized

by these criteria. In some cases, the objectives look forward to techniques described later in this work.

Criterion	Measure	Objective
Performance	Data compression speed	Tune compression algorithm for higher speeds when file size is relatively small.
	Hash algorithm speed	Persist constants used by hashing algorithms.
		Optimize buffering to minimize calls to transformation functions.
		Use loop-unrolling in hashing algorithms where feasible.
	Asymmetric encryption speed	Persist intermediate computed values that are invariant for each signing key.
		Consolidate multiple exponentiation operations that use the same key material.
	Symmetric (bulk) encryption speed	Compress data prior to bulk encryption.
	Speed of combined data delivery and reception operations	Combine compression, hashing, encryption and encoding algorithms to minimize data copying.
		Reuse transport protocol connections where allowed.
Scalability	Maximum end-points supported.	Minimize storage required to maintain end-point parameters.
		Effectively index lookup routines to quickly retrieve end-point security credentials.
	Maximum concurrent transfers supported	Minimize memory required for each transfer.
		Minimize disk usage during data delivery and reception.
	Data compression ratio	Tune compression algorithm for maximum compression rate when file size is relatively large.
	Maximum file size supported	Perform data delivery and reception operations on data in a single pass while reading from media or network.
		Support parallel (multi-connection) delivery.
Security	Private-key storage security	Use encrypted storage to store private keys.
		Require quorum-based techniques to update private keys.
		Restrict access to export private keys.
		Permit only encrypted export of private keys.
	Access Control	Limit application function based on user and group identity and role associations.
		Audit and report access and usage history.
	Data Protection	Employ best-coding-practices for security including memory zeroing, executing sensitive functions at higher privilege levels, etc.
		Isolate intra-tenant processing
	Incident Reporting	Automatically detect and report code and/or data corruption.
		Automatically detect and report unauthorized access.
Reliability	Algorithm selection	Use FIPS approved cryptographic algorithm implementations.
	Fault tolerance	Provide component redundancy to survive device failure.
		Provide automatic retry of operations subject to intermittent resource unavailability.
	Attack resistance	Implement sufficient input checking.
	Delivery	Support check-point restart capability.

Criterion	Measure	Objective
Portability	Supported operating environments	Implement logic in a language guaranteed to provide equivalent functionality across platforms.
		Avoid code dependencies on specific operating environments.
Maintainability	Ease of Use	Provide ample user and technical reference documentation.
		Limit complexity of the user interface.
	Ease of Update	Implement automated update processes.
		Support automated certificate exchange.
		Support secure re-encryption of data when keys expire or must be rotated.
Interoperability	API Support	Use well-known algorithms and message packaging protocols.
		Adopt standard protocols for providing API access to functions.
	Protocol Support	Implement protocol-neutral algorithms where possible.
	Standards Compliance	Publish third-party validation or certification results.

TABLE 4. Architecture Objectives.

3. A Multitenant MFT Architecture.

3.1. SECURITY ALGORITHM BASICS.

Before describing an optimized multitenant MFT architecture, we first present descriptions of fundamental security algorithms, establishing terminology that is used extensively in later sections of this work. Understanding these basic cryptographic techniques is essential in order to grasp the optimization concepts presented later in this work. The basic security algorithms introduced below are presented in the order they are often applied when delivering and receiving secure data.

3.1.1. *Compressing Data.*

Data compression algorithms are useful in managed secure file transfer solutions because they reduce the amount of data that must be encrypted and sent. Also, data compression eliminates recognizable patterns and the alters the frequency of characters thus making some cryptanalysis techniques less effective.

The internals of data compression are outside the scope of this work. However, for our purposes it is important to note that compression functions generally output data at a frequency that is dependent on both the size and the nature of the input. Therefore, it is difficult to predict how much data will be output by a compression function.

3.1.2. *Computing a Message Digest.*

A message digest is a relatively short series of bits[2] generated by a hashing function using a message as input data. The message digest represents a message such that the same hashing function would be reasonably certain to compute a different digest even if only one bit of the input message is altered. There is no guarantee that a random altering of bits in a message would not produce a new message over which the same message digest would be computed. The hash algorithm is only designed to make this possibility extremely remote. In practice, random alterations of message bits are extremely rare. Malicious, intentional, alterations of a message typically are very selective – altering a key term or value but generally leaving the form of the message intact. Although two or more different strings of bits of a given length may be found to have the same message digest, it is extremely unlikely that to two similar messages having only selected subsets if bits altered would have the same digest.

When preparing a message for secure transport, usually the data that is hashed consists of both the message content and possibly some preliminary meta-data. This message digest is then encrypted with the sender's private-key to produce a digital signature. The common hashing algorithms are stateful, allowing a large message to be hashed with a series of calls to the hash algorithm. A finalization call will produce the output message digest.

3.1.3. *Creating and Verifying a Digital Signature.*

As noted above, the message digest produced by a hashing algorithm is encrypted with the signer's private key. The encrypted output forms the basis of the digital signature. The encryption used to create a digital signature is *asymmetric*. The encrypted data produced by using the sender's private key can only be decrypted by using the signer's public key. Therefore, a recipient of the signature must either possess a copy of the signer's certificate before the signature can be verified (decrypted), or the signer must include a copy of his signing certificate with the message.

3.1.4. *Bulk Message Encryption and Decryption.*

While the RSA algorithm illustrated above is effective for encrypting relatively small amounts if data, such as a message digest, it is less effective for encrypting large amounts of data due to its relative slowness. Bulk encryption, that is, encrypting complete messages, is performed using faster, symmetric, algorithms such as AES or Triple-DES. These algorithms are called symmetric because the same key is used both to encrypt and to decrypt data. The disadvantage to symmetric encryption is that the key must either be negotiated by both sender and receiver ahead of time (or during session-establishment as

[2] The SHA-1 hash algorithm produces a 160-bit message digest. The MD5 algorithm produces a 128-bit message digest.

with SSL) or the key must be actually sent along with the data. Obviously, sending an encryption key along with the encrypted data wouldn't be very secure unless the key itself was encrypted. This key-encryption is precisely what happens, therefore, in such secure messaging standards as S/MIME, which underlies popular secure file transfer protocols such as [AS2].

Symmetric encryption algorithms are fast because they apply non-mathematical operations to small sets of data, usually 8 or 16 bytes at a time. The encryptions consist of complex combinations of logical operations like exclusive OR'ing, bit-rotations and table-based substitutions. While fast, symmetric encryption is vulnerable to cryptanalysis if each block is encrypted independently, since the symmetric algorithm will always produce the same cipher-text output given the same clear-text input and key. Therefore, methods of chaining the data are use that combine the output of one block encryption with the input for the next block encryption.

3.1.5. *Encrypting and Decrypting a Symmetric Key*.

If a symmetric key is transmitted along with encrypted data it is first encrypted with the public key of the recipient using an asymmetric algorithm like RSA as described above. This way only the holder of the corresponding private key can decrypt the symmetric key and, thereafter, decrypt the entire message. Any one who has another's public-key certificate can use the public key within the certificate to encrypt a message that only the certificate holder can decrypt.

In a multitenant architecture, several local "tenants" could be expected to sign and encrypt data, possibly to many remote "partners" concurrently. Optimization opportunities here follow two dimensions. First, every file being sent from the same sender can be signed using the same private key. Intermediate computed values based on this key can be shared for every message being sent. Secondly, every message going to the same partner can be encrypted using the same public key. Intermediate computed values based on this public key can be shared also.

3.2. THE SECURE TRANSFORM CELL.

Having reviewed the actual operations that are performed to securely transfer files, this section introduces the concept of the Secure Transform Cell, (the "Cell"), a close relationship of algorithms and data buffers designed to perform the cryptographic and related functions that occur within a Security Context, as described in the preceding sections.

Following a biological metaphor, the Cell is *specialized* to perform transformations of data depending on the requirements of the security context. In this case, the basic transformational building blocks have been arranged to transform the input stream of content into an S/MIME enveloped-data message body prepared for

transport over HTTP and including within it a MIME part containing compressed content and a digital signature. This complex structure corresponds to one permutation defined in [AS2]. However, this elaborate structure should be considered only an example of how various composition or decomposition structures can be deployed within the Transform Cell. Other possible structures may include a corresponding AS2 decomposition structure or an SFTP client session. The most important point to draw out here is that the

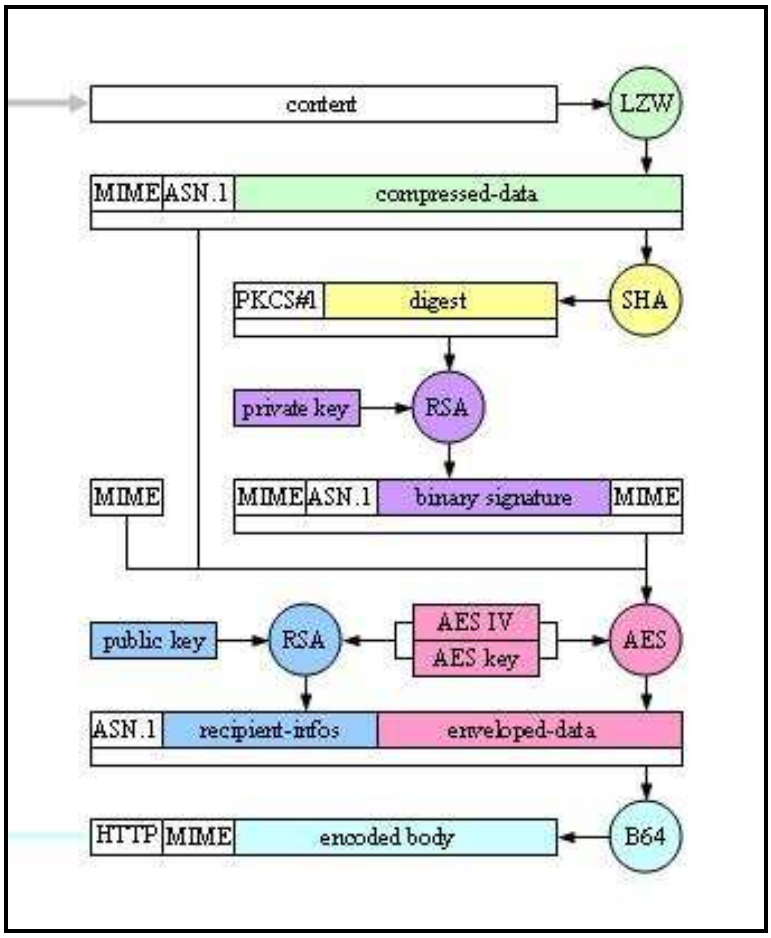

FIG. 4. A Secure Transfer Cell for AS2.

information present in the Transform Cell may either be unique to this cell, as in the case, possibly, of a unique symmetric encryption key for AES, and shared parameters such as the public key used to encrypt the symmetric encryption key. This public key would be the same as that used within all Transform Cells encrypting data intended for the trading partner who issued the certificate containing that public key. Also the private key used to encrypt the message digest, thereby creating the digital signature, would be shared by all Transform Cells signing data on behalf of the sending tenant. The particular algorithms used in this example (LZW compression, SHA message digest computation, base-64 encoding) are not important. Other algorithms could be substituted. Standards adopted by tenants and trading partners or service-level agreements may determine which

algorithms are used. Therefore, an important aspect of the Transform Cell is that it be configurable.

Although no logic within any one Cell is aware of it, many pieces of information used by the Cell are potentially being shared by other Cells. These items include security credentials (certificates and keys), pre-computed values based on these credentials, and configuration parameters that express rules about how to send data to partners or what to do with data received from a particular partner.

Having information used in the Cell that may be concurrently used by other Cells may tempt us to want to define a shared storage area for these values that all Cells can read. However, this is not an optimal approach, especially if mutually-exclusive access to such variables is required. Introducing locking mechanisms to serialize access to shared values between threads will degrade overall performance of a system. Instead, our approach recommends that each Cell be provided it's own independent working set of parameters that completely defines what work it must perform. This approach allows each cell to function independently and, for enhanced security, even in isolation.

3.3. DATA TRANSFORM STREAMS.

In a multitenant MFT solution, processing many concurrent secure file transfer operations places a heavy demand on CPU and memory resources. These demands are exacerbated when transforming and transferring very large files. It is not unusual to find needs for encryption solutions to handle single data objects in the order of several Gigabytes per "file" or "document". Therefore, when we contemplate a truly scalable and robust multitenant MFT architecture, we must identify optimizations to accelerate the flow of data through the Cell while keeping our per-transaction use of resources low. Contemplating a multitenant solution that transfers logical data "files" of this magnitude can be challenging if we began with the assumption that the complex transformations required would be applied piecemeal, on the entire large file, one after another. However, by adopting a streaming architecture, wherein all transformations are applied to a document concurrently, the solution can adopt a much more stable, efficient and secure memory and disk utilization strategy. The structure of the Cell in Figure 4 above takes into consideration the requirement for maintaining low memory utilization. Each of the algorithms used in the Cell is capable of operating "block-by-block" instead of working on the entire file at once. This is a critical aspect of a transformation algorithm – it's ability to accept a potentially very long stream of data in reasonably-sized units or "blocks". This ability allows several independent algorithms to work together, as Figure 4 suggests, processing streams of data continuously, without requiring costly intermediate writes and reads to and from storage media.

3.4. THE MULTITENANCY RESOURCE CUBE.

As we have determined, we can anticipate that in a multitenant environment, many independent Cells will be concurrently processing data, either preparing data for transport or, alternately, receiving secured data for local storage. The need for scalability presents our architecture with restrictive resource limitations on a Cell. The following diagram, Figure 5, depicts several Secure Transform Cells sharing security credentials for several tenants and several partners. This three-dimensional view presents what we refer to as the Multitenancy Resource Cube.

As we saw above in Figure 3, tenant security credentials may be applied along one dimension, shared by each security context requiring a digital signature of the same tenant. This is the "horizontal" or "width" dimension of the Cube. Likewise, partner security credentials are applied along another dimension by each security context requiring the public certificate of the same partner. This is the "vertical" or "height" dimension. Finally, the same intersection of private and public credentials may be concurrent if there are several files being transferred between the same tenant and partner. Along this "depth" dimension, both the private and public credentials are shared.

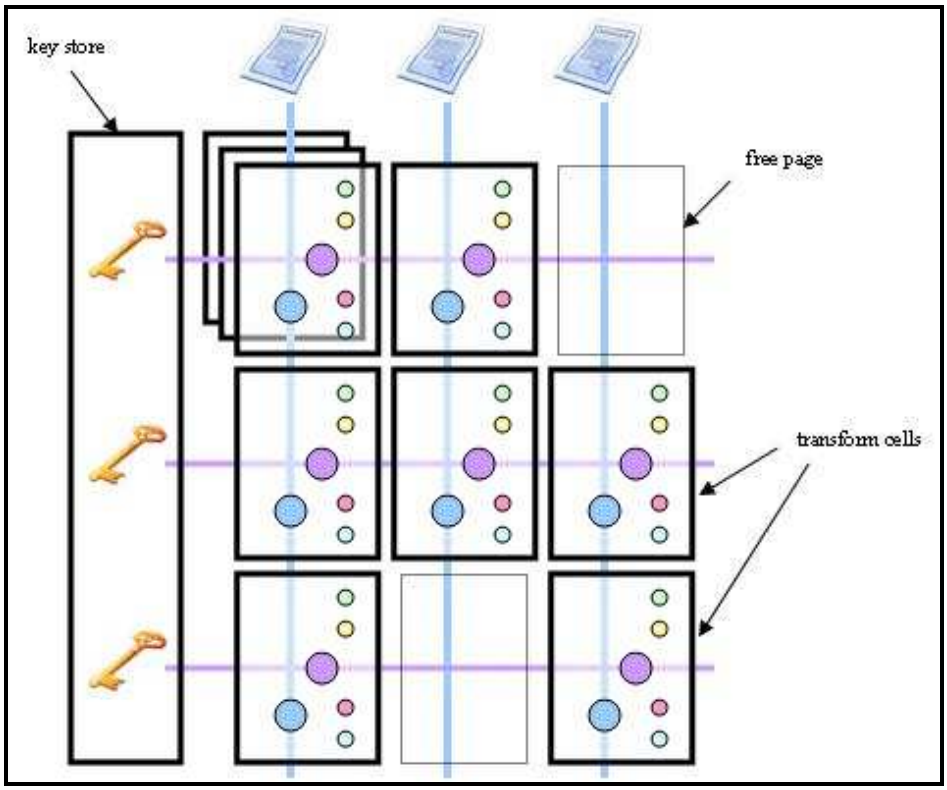

FIG. 5. The Multitenancy Resource Cube.

During the life-cycle of a secure file transfer, several initializing and finalizing operations occur, in addition to the actual transformation of data. For example, if the system must securely send data from Tenant A to Partner 2, the security credentials must be accessed, possibly from an external hardware security module (HSM), algorithms must be initialized and, depending on the algorithms, intermediate or pre-computed values must be prepared. After each use of the security credentials, sensitive data areas must be securely cleared, or "zeroed."

The organization of the Multitenancy Resource Cube, or the "Cube", is conceptually similar to a virtual memory system, consisting of numerous "pages" which are swapped between physical and virtual memory to extend the apparent, or virtual, size of a system beyond its actual physical limits. In the case of the Multitenant MFT system, the page-swapping concept is closely analogous. In our system, the intersection of private and public security credentials is both (a) isolated and (b) frequently reused. By implementing a secure "paging" scenario, the pre-allocated, pre-computed security context required to transform data between any given tenant and partner can be persisted and quickly read into physical memory to instantly begin its transformative operations.

3.5. ALGORITHM OPTIMIZATION EXAMPLE.

3.5.1. *Selection Criteria*.

Thus far, we have identified several high-level concepts for introducing MFT optimization in a multitenant environment. These have focused on (a) the reuse of security credential resources, (b) the close-association of algorithms operating on data streams and (c) the persistence of preconfigured security contexts. Our final area of exploration is in actual algorithm optimization. We identify one particular algorithm that meets several selection criteria: (a) it is used in all security contexts, (b) it is dependent on intermediate computed values that can be persisted (c) it uses values shared across security contexts in a security domain and (d) it is computationally expensive to perform. Using these four criteria allows us to identify algorithms that can reasonably be expected to provide significant performance improvement through optimization. Our example focuses on the RSA algorithm, concentrating on optimizing the modular exponentiation function.

3.5.2. *Modular Exponentiation*.

In many conceivable security contexts, MFT will apply the RSA or a similar algorithm to perform the necessary asymmetric encryption for both digital signatures and for encrypting symmetric bulk-data encryption keys. RSA uses a mathematical operation known as modular exponentiation (*ME*) to compute encrypted output. Algorithms that perform *ME* are computationally expensive. Therefore, opportunities to optimize *ME* computations are very helpful in reducing CPU utilization.

A modular exponentiation consists in raising a number to a certain power (the *exponent*) and then reducing that intermediate result by dividing it by another number and keeping the remainder (the *modulus*). If certain values are used for the exponents and modulus, a cryptographically useful relationship between these numbers exists, where any message *M* raised to the power of the public exponent *E*, modulo *N* produces cipher text that can only be decrypted by raising this value to the power of the private exponent *D*, modulo *N*, provided that *M*, *E* and *D* are each smaller than *N*. In other words, expressed as equations,

```
C = M ^ E MOD N
M = C ^ D MOD N
```

The Public Key Infrastructure (PKI) standards call, then, for a public key to contain the public exponent *E* and the common modulus *N*. The corresponding private key would contain the private exponent *D* and the common modules *N*.

This basic principle of asymmetric encryption can be simply shown by using very small values for *E*, *D* and *N*. Given $E = 5$, $D = 29$ and $N = 35$, we can test asymmetric encryption with several test values, $M_1 = 9$, $M_2 = 11$, and $M_3 = 17$.[3]

```
M1:  C = 9 ^  5 MOD 35 = 59049                                    MOD 35 = 4
     M = 4 ^ 29 MOD 35 = 288230376151711744                       MOD 35 = 9

M2:  C = 11 ^  5 MOD 35= 161051                                   MOD 35 = 16
     M = 16 ^ 29 MOD 35= 8.3076749736557242056487941267522e+34MOD 35 = 11

M3:  C = 17 ^  5 MOD 35= 1419857                                  MOD 35 = 12
     M = 12 ^ 29 MOD 35= 19781359483314150527412524285952     MOD 35 = 17
```

While useful for explanatory purposes, using such small numbers for RSA parameters is not very realistic for two reasons. Firstly, in our example above, only messages smaller in bits than the number 35 (10011b, or 5 bits) could be encrypted and, secondly, with so few possible alternatives for the parameters it would take relatively little time to conduct a successful brute-force cryptanalytic attack. So, in practice, the values for *D* and *N* typically have 1024, 2048 or 4096 bits, making a trial-and-error approach to factoring an encrypted message rather more laborious.

3.5.3. *Modular Exponentiation Algorithm*.

Figure 6 presents an efficient implementation of modular exponentiation using the Montgomery method for modular multiplication.[4] The message *M*, which in the case of a digital signature operation would be the message digest formatted according to PKCS#1,[5] is raised to the power of the private key exponent *D*, modulo the public exponent *N* to produce the output cipher-text *C*.

---

[3] Note that the numbers selected for RSA parameters cannot be chosen arbitrarily. They must possess a particular mathematical relationship to each other that, although fascinating to explore, is beyond the scope of this work.

[4] [HAC] p. 620, algorithm 14.94.

[5] [PKCS1] pp. 27, 28..

Input:	N	$= (N_{l-1} \ldots N_0)_b$
	R	$= b^l$
	n'	$= -N_0^{-1} \bmod b$
	D	$= (D_t \ldots D_0)_2$ with $D_t = 1$
	M	= integer, where $1 \leq M < N$.
Output:	$M^D \bmod N$.	
1.	$U \leftarrow$ MontMul(M, $R^2 \bmod N$)	
	$C \leftarrow R \bmod N$.	
2.	For i from t down to 0 do:	
	2.1	$C \leftarrow$ MontPow2().
	2.2	If $D_i = 1$ then $C \leftarrow$ MontMul(C, U).
3.	$C \leftarrow$ MontMul(C, 1)	
4.	Return(C).	

FIG. 6. Montgomery Modular Exponentiation.

In a multitenant system, the private key for a tenant may be reused for every signature created for the tenant and every asymmetric decryption of a message received for the tenant. Given persistent values, then, for N, and D, we can pre-compute R, R^2 mod N, n', t, and the initial value for C computed in 2.1. R is a power of two derived directly from the size of N. R^2 mod N is reused once for each operation to compute the initial value of U. n' is the reciprocal of the lowest-order word of N. Its use is not evident in Figure 6, but it is a multiplier within the function MontMul(X, Y). We provide optimization techniques for computing m' in [MINV]. The loop in step 2 counts from t down to zero, t representing the highest-order non-zero bit of D. t is fixed as it is derived directly from the size of D. Finally, the initial Montgomery product in 2.1 is a squaring of R mod N, so it is constant for each N.

Having defined values that can be pre-computed for each use of the tenant private key (D,N), Figure 7 illustrates two sections of code. The first performs pre-computation of the reusable values. This code should execute when the security context is created in the page. The second section of code performs the *ME* using the pre-computed values.[6]

When the security context is loaded into a physical memory page, reused values are pre-computed as shown in OnLoadPrecompute. Thereafter, for every *ME* operation performed using the private key parameters, only the code in OnSign needs to be executed.

[6] In practice, efficient computation of modular exponentiation as part of the RSA algorithm actually involves several distinct exponentiations using derivatives of the private exponent.

```
//SecurityContext::OnLoadPrecompute
//Input:    N          = tenant modulus
//          D          = tenant private exponent
//Output:   Q          = R mod N
//          R          = R^2 mod N
//          N0'        = -N0^-1 mod b
//          t          = high-order bit# of N, base zero

w = N.Words();
R.Zero();
R[w] = 1;
Q = R.Mod( N );
w <<=1;
R.Zero();
R[w] = 1;
R.Mod( N );
ni = MINV( N[0] );
t = D.HiBit();

//SecurityContext::OnSign
//Input:    M          = PKCS#1 digest
//          Q          = R mod N
//          R          = R2 mod N
//          t          = high-order bit# of N, base zero
//Output:   C          = encrypted digest

U = M.MontMul( R );                          // uses N0'
C = Q;
for( i = t; t; t--) {
          C.MontPow2();                      // uses N0'
          if( D.Bit( i ))
                    C.MontMul( U );          // uses N0'
}
C.MontMul( 1 );                              // uses N0'
```

FIG. 7. Pre-computation and Signature.

4. Summary.

A multitenant architecture for managed, secure file transfer can be designed to scale well both in terms of performance and resource utilization. As we have described above, performance can be optimized by identifying reusable intermediate values and pre-computing them upon creation of a security context, representing the intersection of tenant and partner security parameters. High levels of scalability can be obtained by restricting the size of per-transaction memory and disk utilization. Processing data as a continuous stream, applying several transforming operations concurrently, allows for the creation of "secure virtual memory" pages and efficient loading of pre-configured security contexts using familiar LRU (least recently used) types of algorithms. Future works will explore efficient implementation of these concepts.

5. References.

[AES] National Institute of Standards and Technology (NIST), *Advanced Encryption Standard (AES)*, Federal Information Processing Standard (FIPS) Publication 197, November 26, 2001.

[AS2] Moberg, D., R. Drummond, *Mime-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2)*, RFC 4130, 2005.

[DES] National Institute of Standards and Technology (NIST), *Data Encryption Standard (DES)*, Federal Information Processing Standard (FIPS) Publication 46-2, December 30, 1993.

[GISS] Paul van Kessel, et al., *11th Annual Global Information Security Survey (GISS)*, Earnst & Young, October, 2008.

[HAC] Menezes, A., P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[MONT] Montgomery, P. L., *Multiplication Without Trial Division*, Mathematics of Computation, Vol. 44, pp. 519-521, 1985.

[MINV] Walling, D., *Optimizing Modular Inverse Computation*, 2007.

[PKCS1] *PKCS #1 v2.1: RSA Cryptography Standard*, RSA Laboratories, June 14, 2002.

[SHA] National Institute of Standards and Technology (NIST), *Secure Hash Standard (SHA)*, April 17, 1995.

Glossary of Abbreviations

AES	Advanced Encryption Standard
AS2	Mime-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2
HSM	Hardware Security Module
LZW	Lempl-Ziv-Welch Compression Algorithm
SaaS	Software as a Service
MFT	Managed File Transfer
MIME	Multipurpose Internet Mail Extensions
PKCS	Public-Key Cryptography Standards
PKI	Public Key Infrastructure
RSA	Rivest, Shamir, Aldeman Algorithm
S/MIME	Secure MIME
SHA	Secure Hash Algorithm
SFT	Secure File Transfer

Index

A

Access Control, 3
AES, 12
AS1, 13
AS2, 13, 14
Asymmetric Decryption, 19
Asymmetric Encryption, 12
Authentication, 3

C

Capacity Metrics, 4, 5
Certificate, 4
Compression, 11
Cost Savings, 4
Cryptography, 3

D

Digital Certificate, 5, 6, 12
Digital Signature, 4
Disaster Recovery, 3

E

Electronic Commerce, 3
Encryption, 4, 8

H

Hardware Security Module, 17
Hash Function, 12

I

Information Security, 3
Integrity (Data), 3

K

Key Management, 5, 8

L

LRU (algorithm), 20

M

Managed File Transfer, 4, 6, 8, 11, 15, 17
Managed Service, 3, 4
Message Digest, 4, 12
Modular Exponentiation, 17, 18
Montgomery Product, 18, 19
Multitenancy, 6, 11, 16, 17, 19, 20
Multitenancy Resource Cube, 17

N

Non-Repudiation, 3

O

Optimization, 4, 7
Optimzation, 13

P

Privacy, 3
Private Key, 4, 13
Public Key, 5, 12
Public Key Cryptographic Standards, 18
Public Key Infrastructure, 18

R

RSA, 12, 13, 17, 18

S

S/MIME, 13
Scalability, 20
Secure File Transfer, 3, 4, 8, 17, 20
Secure Transform Cell, 13, 16
Security Context, 6, 13, 19, 20
Security Credentials, 6
Security Policy, 8
Signature Verification, 4
Signed Receipt, 4
Software as a Service, 4, 6
SSL, 13
Symmetric Encryption, 13
Symmetric Key, 5, 13

T

Transfer Domain, 6
Triple-DES, 12