Adapting Encrypted Data Streams in Open Architectures

Peter Reiher, Kevin Eustice, Kai-Min Sung
File Mobility Group
Computer Science Department
University of California, Los Angeles
{reiher,kfe,kaimin}@cs.ucla.edu

Abstract

Open architectures meet the demands of increasingly complex and diverse networks by adding programmability to some or all components of the network infrastructure. The effectiveness of this programmability, however, is severely reduced in the domain of encrypted data streams. Because encrypted streams are opaque, existing solutions to adapting encrypted data are limited to either performing link-level encryption across each node, or restricting adaptations to those that do not require knowledge of the content. These restrictions either reduce the functionality of the system or compromise the security of the data stream. This research examines an alternative method of adapting encrypted streams through intelligent tagging and layering of the data content. Sending separately encrypted layers of the data allows for adaptation of the data stream without actually decrypting the data along any point of the connection.

1. Introduction

Open architectures are increasingly recognized as a promising way to handle the growing complexity of networks and communications applications. However, open architectures pose many unsolved problems. One such problem is providing open architecture benefits while simultaneously providing data transmission security.

The most popular method of providing such security is encryption, which renders data streams opaque to potential attackers. Ideally, data encryption should convert the contents of a data transmission into seemingly random bits that contain no hint of the original data's values. Only those sharing the secret keys are able to convert the encrypted data back to its original form. Standard cryptography thus works hard to hide all clues about the contents of a data packet, since even the slightest knowledge might help an attacker guess the contents. To the extent that cryptography succeeds, an attacker cannot read or alter the data stream. Unfortunately, beneficial adaptive middleware also cannot compress, filter, or otherwise operate on the stream, except in ways that are totally independent of its content.

Three approaches are commonly used to solve this problem:

- Allow open-architecture nodes in the middle of the network to decrypt the data, adapt it, and re-encrypt the adapted version before sending it on its way.
- Limit internal open-architecture nodes to adaptations that do not require knowledge of data content.
- Perform all adaptations end-to-end, ensuring that all content-specific adaptations are performed before encryption [Noble00].

The first approach requires knowledge of which middleware sites are trustworthy. It also requires secure key distribution, either ahead of time or on demand, which usually implies a trusted infrastructure. No such infrastructure currently exists, and creating one is challenging. Also, performing initial key distribution securely adds to complexity and startup delay, and multiple encryption and decryption steps per data unit can be very expensive and slow the data transmission.

The second approach severely limits the benefits the adaptive middleware can provide. Some adaptations (such as adding forward error correction or other reliable delivery information) can be performed on the encrypted form of the data, but this approach rules out the majority of adaptations, even the extremely common adaptation of performing lossless compression. In many cases, some content-aware adaptation is required to allow useful delivery of the data.

The third solution is the one most commonly used in practice, but this solution is also limiting. Performing adaptations only end-to-end reduces the agility of the system, since all possible transformations must be performed before the data is sent. If a link in the middle of the network is sometimes heavily congested, the data must be pre-adapted for the congested condition, even though the link may have greatly improved by the time the packet actually arrives. In the case of a multicast transmission, the quality of the stream must be adapted to the least capable link, and the remaining destinations, which might very well be capable of receiving the full stream, must accept the degraded stream. This solution, in essence, implies that the open architecture is limited to the end nodes.

2. Adapting Encrypted Data

The existing approaches trade off the performance of the system and the power of the open architecture to obtain high security. They treat the degree of security provided by the cryptography as being sacrosanct. However, in practice very few security failures arise from breaking a system's cryptography. Stealing keys, hacking into systems, or otherwise working around the cryptography are far more common, because the cryptographic algorithms available today are sufficiently strong to ensure that breaking them is expensive. Since these algorithms are actually one of the strongest parts of the system today, one might consider trading some of their strength to deal with areas of the system that are less well handled.

Company News Brief to Directors

<Confidential>
Today, a new product – the widget – was announced. Widget production will commence 2Q 2002.
</Confidential>
Our stock was up 3 ½ points today after our earnings announcement and conference call.

<Secret> President Rivest announced today his plans to retire. Vice-President Shamir will assume control of the company on the 20th. This information should be considered company private.
</Secret>

Tomorrow's lunch will be roast beef and cabbage.

Figure 1. Full text of document (with security tags)

We are investigating this approach by researching adapting encrypted data streams in open architectures. In effect, with a small amount of preparation on the sending node and some extra information accompanying the encrypted packets, some forms of encrypted data can be adapted on the basis of their content. At the simplest level, this could be the attaching of a tag identifying the priority of each piece of data, and letting routing nodes drop low priority data if necessary. Alternative adaptations could include caching of important parts of the encrypted video, or alternative routing for high-priority sections. This approach definitely compromises the degree of protection offered by the cryptographic system, but we believe that often even a weaker cryptographic system is harder to attack than other components involved in the transmission.

This method implies that the data stream needs to be broken up into discrete segments, appropriate for tagging. In some cases, this can be done fairly easily. Consider a web document tagged with different security requirements on different paragraphs (Figure 1). Because at least a

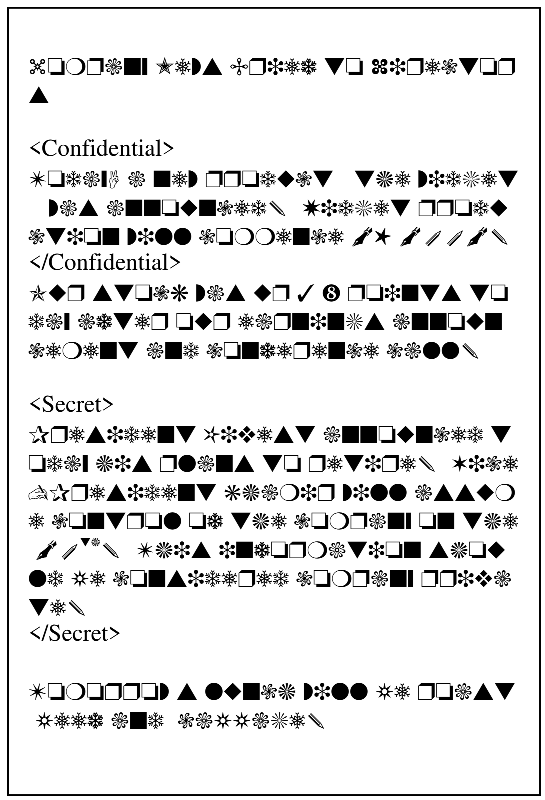

Figure 2. Encrypted Document with tags

portion of the document is sensitive, it must be transmitted in encrypted form. But perhaps the security policy of the owner also requires that the most secure portions of the document should never be sent over a wireless link, even though they are encrypted. Unless source routing or some other method of specifying a particular route through a network is used, the route is not determined until the last instant. Whether a transmission will cross a wireless link is only known immediately before the link is crossed, since alterations in routing might occur at any moment. When transmitting such documents across the network, the sender would have to filter out the sensitive paragraphs first, to be safe. However, if one separately encrypts and tags the differing paragraphs at the source end (Figure 2), the open architecture can check at each outgoing link whether a wireless network is about to be used. If it is, sensitive material can be dropped, while still allowing the remaining material to be properly decrypted at the other end (Figure 3).

Company News Brief

Our stock was up 3 ½ points today after our earnings announcement and conference call.

Tomorrow's lunch will be roast beef and cabbage.

Figure 3. Decrypted text after filtering

For a video transmission, the system must intelligently segment the data stream. An image can be transmitted in a high-resolution form or a low-resolution form, the latter requiring far fewer bits to be sent than the former. Converting from the high-resolution form to the low-resolution form is relatively easy. If the transmission encounters a congested link along the way, perhaps the destination user would prefer quick delivery of the low-resolution image to slow delivery of the high-resolution image. If the security conditions require encryption of the data, the sender can convert the high-resolution version to the low-resolution form, encrypt a copy of each separately, and tag them as "high-resolution" and "low-resolution." An adapter placed on a network node situated close to the possibly congested link can periodically examine the conditions of the link. When the two encrypted versions of the data arrive, the adapter can choose which version to send on, based on current link conditions, without requiring decryption.

This method is simple, but is not efficient or appropriate in every situation. If the network is not congested, traffic is increased unnecessarily by the size of the low-resolution version of the image. Additionally, this solution limits our adaptation to just that of dropping the high-resolution version. If instead we wished to drop out the color components, or decrease the frame rate, we would be unable to do so. However, there is a less costly and more flexible alternative.

3. Layered Encodings

By employing a format in which the data is encoded in a composable (or layered) fashion, most redundancy can be eliminated. Layered encodings break the data stream into an ordered collection of layers. The base layer provides a low-quality but complete version of the stream, and each subsequent layer provides a high-quality version of the stream when composed with the lower layers. In this manner, we eliminate the overhead of sending multiple separate and complete data streams, and gain flexibility in the kinds of adaptations that we can perform.

In the case of our video transmission, composing additional layers with the low-resolution data stream creates the high-resolution version. The combined size of both is comparable to the size of the final high-resolution version, and there is little or no wasted bandwidth when adaptation is not required. Additional flexibility is gained as we can select alternatively to drop out color components but still deliver a high-resolution image—whereas before all we could drop was an individual complete frame. Other adaptations are now possible as well, such as best-effort delivery of nonessential components and reliable delivery of essential components. Many data encodings support this type of layering, including layered Robust Audio [RobustAudio], DCT, MPEG-2, H.263+, WaveVideo [Dasen96].

4. Cipher Selection

Another example of encryption for adaptation occurs in stream cipher encryption. Block ciphers are easy to use for the above encryption, however these ciphers are not necessarily suited to variable rate data flows with real-time requirements. Block ciphers are slow, since they must wait for an entire block of plaintext before encrypting. Additionally, error propagation is high—a single-bit error will affect the transformation of all other symbols in the plaintext block. Stream ciphers, however, alleviate this problem by encrypting each symbol independently, without regard to the remaining symbols.

A stream cipher, in which the next-state function is determined by a portion of the ciphertext from the last transform, does not necessarily fit easily into an adaptive world. When packets can be dropped arbitrarily, the key state on the decrypting side will not necessarily match the key state on the encrypting side, and the data packets will not be correctly decrypted. However, a stream cipher operating in output-feedback mode (such as RC4), where the next-state function is based solely on the current state, can be successfully used by implementing a system that passes along metadata that tracks the changes to the data stream. This can be done easily by tracking dropped packets and adding a tag to delivered packets that records the number of skipped packets. When the packet is received, the correct number of placebo packets is inserted into the stream to synchronize the key state. When implemented efficiently, stream ciphers can be significantly faster than block ciphers. [Schneier00]

Additionally, this speed-up should offer a significant advantage given the performance-related requirements of adaptive data streams. Most notably, when there is a large number of components to the data, such as a layered encoding, avoiding a heavy block cipher per-packet penalty is advantageous.

5. Research Directions

We have designed and built two sets of encryption adapters designed to work with wavelet and MPEG encoded video, each with a slightly different methodology. This work indicates some possibilities for adapting encrypted data, but there are significant research issues to be investigated.

First, there is an obvious question of how widely applicable these techniques for adapting tagged encrypted data will be. Not all data formats can be handled in this way. As the example of a document with varying security levels shows, the basic idea can be applied to data formats that do not have the high-resolution/low-resolution character, but it is not yet clear how many formats are amenable to clever pre-adaptations and encodings. A second concern is that dividing the data into separately encrypted segments and tagging each in plaintext will leak some information about the content. For example, knowing that a particular document contains sensitive sections will be helpful to an eavesdropper. In general, this approach exchanges some knowledge of the underlying data format for added flexibility and better performance. This tradeoff is inherent to this mode of adaptation.

A third possible concern relates to whether the open-architecture nodes handling the encrypted message are trustworthy enough to be allowed to perform adaptations on the encrypted data. In some scenarios, they handle encrypted data because we will not trust them with the plaintext form. If they are not trusted with the plaintext, should they be trusted to perform any manipulations on the ciphertext, even those that are sometimes acceptable? Perhaps the open architecture can include features that allow applications to force their data flows to visit only trusted adaptation nodes.

Another issue is not a concern, but an interesting theoretical generalization. Consider the following possibility: Given a plaintext P, key K, encryption algorithm E, and a pre-defined adaptation function F, is it possible to create a system that produces

$E(P,K, F) = \{C, F'\}$

such that

$F(P) = P'$

and

$E(P',K) = C'$

and

$F'(C) = C'$

If it were possible to create such a cryptosystem for arbitrary plaintexts and adaptations that had strong security and if the function F' produced by such a system was reasonably small itself, then F' would be attached to C and the result could be sent through the open architecture. If a problem requiring the adaptation F was encountered, F' would be applied to C, producing C' without ever decrypting the message.

Research in encrypted functions suggests that this type of adaptation may be possible [Sander98]. However, existing work in this area is highly theoretical, and is thus far confined to realm of pure mathematics. Additionally, similar to the approaches discussed above, information regarding the representation of the underlying data is still exposed.

Finally, based on our preliminary results, it seems clear that performance is an issue in the more flexible case. Further research must be done in cipher selection and development. Replacing our block cipher implementation with an adaptive stream cipher discussed above may provide a substantial performance gain.

Adapting encrypted data is much more limited than adapting unencrypted data. In essence, identification of the data, and preparation for adaptation is required before the data is encrypted. Thus, this method only supports adaptations that can be predicted. Preparing for totally different types of adaptation might also be inefficient. On the other hand, many open architecture systems already require some pre-definition of which operations might be performed on the data before it is sent.

6. Conclusions

Combining open-architecture network capabilities with security requirements has, to this point, been difficult. Adaptations had to be strictly limited, or nodes that might be in the middle of a network had to be highly trusted, or security would be limited. In essence, these choices are the standard cases in securing systems: limiting functionality of the system, imposing significant performance penalties, or not using security. Experience has shown that when such choices must be made, security is usually the first place where compromises are made. If these are the only choices available, either open architectures will not be widely used, or they will often have poor data transmission security.

Fortunately, this research demonstrates another possibility. There are several reasonable approaches for open architectures to perform useful alterations on some forms of encrypted data without decrypting that data. The existence of these approaches offers wider applicability of open architectures. These approaches compromise security to some extent, but not nearly as much as not using cryptography at all, and arguably less than trusting open architecture nodes to operate on unencrypted data.

The breadth of possible adaptations under encryption remains to be discovered. The number of adaptations and data types that can be successfully handled with similar techniques is unclear at this point. We intend to discover as many reasonable adaptations of this kind as we can, and implement a set of the most compelling ones. Research on more general methods of adapting encrypted data, including special cryptographic methods designed for that purpose, would also be valuable.

7. References

[Dasen96] M. Dasen, G. Fankhauser, B. Plattner, "An Error Tolerant, Scalable Video Stream Encoding and Compression for Mobile Computing." *ACTS Mobile Summit 96*, Granada, Spain, November 27-29, 1996.

[Noble00] Brian Noble, "System Support for Mobile, Adaptive Applications." *IEEE Personal Communications* Vol. 7, No. 1, February, 2000

[Panda] PANDA: Peer Agent Negotiation and Deployment of Adapters—http://fmg-www.cs.ucla.edu/panda/.

[RobustAudio] UCB's Robust Audio at ICSI—http://www.icsi.berkeley.edu/raudio/.

[Sander98] T. Sander, and C. Tschudin, "Towards mobile cryptography." *Proceedings of the 1998 IEEE Symposium on Security and Privacy,* Los Alamitos, CA, USA: IEEE Comput. Soc, 1998. p.215-24.

[Schneier00] Bruce Schneier, Crypto-gram, January 15, 2000—http://lwn.net/2000/0120/a/cryptogram.html.

[Tennenhouse96] D.L. Tennenhouse and D.J. Wetherall, "Towards an Active Network Architecture." *Computer Communication Review*, Vol 26, No. 2, April 1996.

[Wetherall98] David J. Wetherall, John Gutag, and David L. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols." *Proceedings of IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[Yarvis99] Mark Yarvis, Peter Reiher, and Gerald J. Popek, "Conductor: A Framework for Distributed Adaptation." *Proc. Seventh Workshop on Hot Topics in Operating Systems (HotOS VII)*, Rio Rico, AZ, March 1999.

[Yarvis00] Mark Yarvis, "Challenges in Distributed Adaptation." Computer Science Colloquium, Harvey Mudd College, March 2, 2000.