Editors: Peter Gutmann, pgut001@cs.auckland.ac.nz
David Naccache, david.naccache@ens.fr
Charles C. Palmer, ccpalmer@us.ibm.com

Room at the Bottom

Authenticated Encryption on Slow Legacy Networks

Sean W. Smith
Dartmouth College

When we hear the term *distributed computing*, it's tempting to think of the things we might do on a recent-issue PC or laptop connected to an ultrafast Internet—surfing the Web, sending documents and images, listening to streaming audio, buying things. So when we think of cryptographic protections on the data exchanged in distributed computing, it's easy to think of the standard protections we use in these scenarios, including public-key handshakes for authentication and establishment of session keys and AES (Advanced Encryption Standard) with ultralong keys running in CBC (cipher block chaining) mode to protect the data from prying eyes. If we're going to be fussy and worry about data integrity as well as confidentiality, we'll add an HMAC (hash-based message authentication code) or such.

But there's more to the story. (Isn't there always?) These scenarios all feature comfortable excess—computers are new and powerful and have ample memory, data transmissions are plump with bytes (so a few more here and there don't matter), the network is fast and has plenty of bandwidth, and users won't notice if data is a bit late now and then.

However, just as life on Earth consists of far more than just sentient mammals, the computing infrastructure consists of far more than just these relatively advanced scenarios. In particular, consider the electric power grid—arguably the world's largest distributed system and the critical infrastructure that underlies all other critical infrastructures. Because electricity can't be stored efficiently in large quantities, grid operation requires maintaining delicate, near-real-time coordination among generation, transmission, and consumption—hence the need for distributed computation and communication. Adversaries forging communications might cause havoc by impersonating the control center and telling a substation to open or close lines incorrectly, or impersonating a substation and lying to the control center about the state of some critical data. (Recall that an alarm system's failure to send out correct alerts exacerbated the 2003 Northeast blackout in the US.)

The consequences of a security problem in this distributed computation might be considerably higher than in personal browsing—and argue for a cryptographic solution. However, the straightforward, textbook-style solution for personal browsing doesn't work here; the legacy grid's computing infrastructure constrains the comfortable excess necessary for this approach:

- Vast portions of the computation infrastructure are old, and won't change anytime soon. We can't begin a solution with "replace it all with the latest and greatest"; we have to work with what's there.
- This infrastructure features very slow (for example, 9600 baud) serial lines and occasionally might operate at capacity. In personal browsing, we can ignore data bloat, but not here.
- Grid control operates in near real time—if a power line has fallen, we can't wait seconds for the communication to finish. We need to think about the latency cost of any protections we add.
- The communications we need to protect might be very short—dozens of bytes, not millions—and might not necessarily announce their lengths a priori.
- Any computing devices we add might also have limited power and memory, and won't likely be replaced anytime soon.

Solving this problem requires stepping away from the picture-perfect world of textbook assumptions and into some dusty corners of cryptography accommodating real-world constraints.

However, in one aspect, our problem is easier than personal browsing: the network topology is fairly static. The correct party on the other end of the line will be one of only a few possible candidates, rather than anyone on the Internet; as a result, we don't need a public-key infrastructure and 100 trust roots to make conclusions about a huge population of legitimate senders.

Bump in the Wire

So what do we do? To begin, we must remember that cryptography isn't just for confidentiality. Often in legacy networks, data content isn't particularly sensitive—what's critical is that a fresh communication comes from the correct sender. This requirement takes us away from the Crypto 101 concept of "you need the secret key to make sense of this message" to the Crypto 102 concept of "you need the secret key to have produced this message in the first place." In the typical model, the sender uses the secret key to produce some cryptographically secure message authentication code (MAC) on the message data, and sends that along with the message.

When random noise can corrupt a message, a standard engineering solution is to append some type of checksum—such as a cyclic redundancy checksum (CRC)—to the message. The checksum function $C()$ is designed so that if random noise changes the message m and/or checksum c to some m',c', it's unlikely that $c' = C(m')$. However, security requires protection against malicious, not random, errors; if adversaries know how to change m to m', and then are able to change the checksum c to one that matches m', we lose. Standard CRC techniques fall prey to this, which is why we need cryptography.

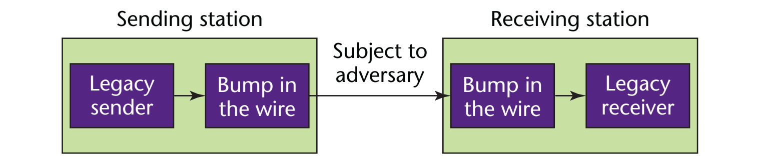

Figure 1. The basic framework adds a bump in the wire at either end. These specialized embedded boxes add cryptographic protection.

We can't replace the legacy network or the legacy endpoint devices. So, we live with them, but insert a *bump in the wire* (BITW) on either end. We use these specialized boxes to add cryptographic protections to the legacy network. Figure 1 sketches this framework.

Of course, now we have to figure out what these protections should be!

A critical challenge is latency. Adding cryptographic integrity protections to communications can't delay the process of sending and receiving a message. Keeping latency down leads to thinking about obvious issues, such as reducing the computation's overhead. However, in the context of our constraints, it also leads to a perfect storm of more subtle issues.

When dealing with slow serial networks, we need to think about bit overhead. The MAC or cryptographic checksum we add must be long enough to prevent adversaries from randomly guessing the correct one. For example, if the MAC value is only 4 bits long, adversaries have a one-in-16 (2^4) chance of finding one that matches the forged message simply by guessing. Consequently, we measure an integrity method's security by how long the MAC value is and thus how resistant it is to guessing—assuming, of course, that the system is designed to be cryptographically strong and resistant to more clever computational attacks.

But because every byte costs a delay in transmission, we want to minimize the number of extra bytes.

With the communicating sender's slowness, we don't have the luxury of waiting for the message to fully arrive before computing our cryptographic protections and sending them on. Thus, instead of the typical offline model, we need to think about an online model. When the bits arrive, we must do something quickly and send them on. Consequently, we probably won't send the MAC until after we send the message, because we won't know what it is until we see the entire message.

For computer scientists, symmetric ciphers are always block ciphers—keyed permutations on bit strings with a fixed block length. Secure operation requires chaining together successive blocks in a message. For short transmissions, we must also consider the rounding up of bits to a full block length. For instance, if our message is only a few bytes, extending it out to a full 128-bit AES block adds significant overhead. Consequently, we need to move from block- to stream-oriented ciphers.

To deal with the receiver's slowness, we also need to think about *message holdback*. Our receiving BITW won't know whether a message is valid or forged until it receives the entire message and the MAC value. If our receiving BITW starts forwarding bits on to the legacy endpoint as soon as it receives them, what should it do if the MAC check fails? It can't reach down the wire and pull the bits back! But the alternative is to buffer the entire message until the MAC arrives and is verified, and

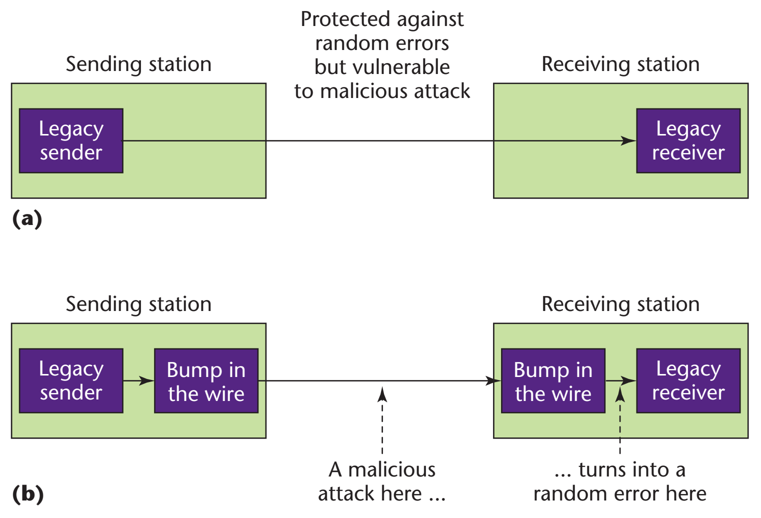

Figure 2. (a) Legacy networks already protect against random errors. (b) By transforming a hash-based message authentication code (HMAC) failure into a random error, the receiving bump in the wire (BITW) can deliberately foul the end of the closing handshake.

then start sending it on. Unfortunately, this alternative approach delays each message by at least the length of time it takes to transmit the message over our slow network and also requires sufficient memory at the receiving BITW to hold the entire transmission.

Stream Ciphers

If we need confidentiality, we can't use a traditional block cipher in a block-oriented fashion; this would require rounding up to a block length before sending. However, standard textbooks give ways to use blessed block algorithms in stream fashion. For example, in the project my lab built in this space, we used AES in counter mode: we used the secret key to transform a sequence of counter values into a pseudorandom sequence, which we could then use as a stream cipher.[1,2] This approach also has the advantage that we can do the hairy computation—the AES operations themselves—independently of the plaintext, before we even know the plaintext bits. The only operation affecting latency is a simple bitwise exclusive-OR (XOR); the same considerations apply at the receiver.

For integrity and authenticity, we can use standard HMAC based on SHA-1, but truncate the output to a value that's long enough to be secure yet smaller than the full 160-bit SHA-1, which takes too long to transmit. The MAC value length has other costs: because the receiving BITW can't know whether the message is valid until it receives the entire MAC value, it can't finish forwarding the message to the receiving endpoint until the entire HMAC has arrived. (Although claiming absolute limits is a risky business, it certainly appears the latency can never be lower than the time it takes to transmit the HMAC.)

The receiving BITW can't forward the message to the receiving endpoint until it has obtained and verified the HMAC. However, it can start transmitting bytes as soon as it gets them (and XORs, if we want confidentiality too). Earlier, I lamented how the BITW can't reach down the wire and pull back the bits if the HMAC fails. However, it doesn't have to. Although these legacy networks weren't designed for security, they were designed to work despite random failures. Consequently, the legacy protocols typically have some way—such as a checksum or a closing handshake—for an endpoint to detect if the message was damaged in transit. Building on this, Andrew Wright and his colleagues made the clever observation that a receiving BITW can transform a malicious error into a random error: if the HMAC value fails, the receiving BITW deliberately fouls the end of the handshake.[3] Figure 2 shows this trick.

Patrick Tsang and I designed and prototyped the YASIR (Yet Another Security Retrofit) protocol, which followed this design sketch[1] and tried to further optimize it by exploiting local knowledge.[2] Other solutions include the Hallmark/SSCP (Secure SCADA Communication Protocol) project developed by Mark Hadley at Pacific Northwest National Lab and commercialized by Schweizer Engineering.[4] The details of Hallmark are still confidential, but its BITW version appears to use message holdback.

In personal browsing scenarios, the notion of data arriving gradually seems rather natural (if annoying). We don't wait until all the gigabytes arrive to suddenly see them; things appear piecemeal. In the slow legacy networks we're considering, latency forces us to process data on the fly.

Surprisingly, symmetric cryptography analysis has focused on offline schemes.[5,6] Online encryption and decryption analysis was slow in coming[7,8] and even then focused on blockwise models rather than the streamwise model forced on us by low-latency applications on slow networks.[9]

Computing systems in the real

world aren't always as simple and conveniently powerful as one might think; engineering cryptography for these systems can be interesting. □

References

1. P. Tsang and S. Smith, "YASIR: A Low-Latency, High-Integrity Security Retrofit for Legacy SCADA Systems," *Proc. IFIP TC 11 23rd Int'l Information Security Conf.*, Springer, 2008, pp. 445–459.
2. R. Solomakhin, P. Tsang, and S.W. Smith, "High Security with Lower Latency in Legacy SCADA," *Proc. 4th Ann. IFIP Working Group 11.10 Int'l Conf. Critical Infrastructure Protection*, Springer, 2010, pp. 63–80.
3. A. Wright, J. Kinast, and J. McCarty, "Low-Latency Cryptographic Protection for SCADA Communications," *Proc. 2nd Int'l Conf. Applied Cryptography and Network Security*, LNCS 3089, Springer, 2004, pp. 263–277.
4. J. Chappell, "Hallmark Cryptographic Serial Communication," 2010; www.oe.energy.gov/DocumentsandMedia/Hallmark_Cryptographic_Serial_Communication.pdf.
5. M. Bellare et al., "A Concrete Security Treatment of Symmetric Encryption," *Proc. IEEE Symp. Foundations of Computer Science* (FOCS 97), IEEE CS Press, 1997, pp. 394–403.
6. P. Rogaway, "Nonce-Based Symmetric Encryption," *Proc. FSE*, LNCS 3017, Springer, 2004, pp. 348–359.
7. M. Bellare, T. Kohno, and C. Namprempre, "Authenticated Encryption in SSH: Provably Fixing the SSH Binary Packet Protocol," *Proc. ACM Conf. Computer and Communications Security*, ACM Press, 2002, pp. 1–11.
8. A. Joux, G. Martinet, and F. Valette, "Blockwise-Adaptive Attackers: Revisiting the (In) Security of Some Provably Secure Encryption Models: CBC, GEM, IACBC," *Crypto*, LNCS 2442, Springer, 2002, pp. 17–30.
9. P. Tsang, R. Solomakhin, and S.W. Smith, *Astro: Authenticated Streamwise On-line Encryption*, tech. report TR2009-640, Computer Science Dept., Dartmouth College, Mar. 2009.

Sean W. Smith *is a professor at Dartmouth College. Contact him at sws@cs.dartmouth.edu.*