Prudent engineering practices for cryptographic protocols

CS380L: Mike Dahlin

November 17, 2004

"The principles are neither necessary nor sufficient for correctness. They are however helpful, in that adherence to them would have prevented a number of published errors."

Abadi and Needham

1 Preliminaries

1.1 Review

1.2 Outline

1.3 Preview

2 Overview of security

- Key lesson: technical solutions alone insufficient; good designer needs to think about the big picture; need to consider how system will be developed, maintained, used
  - Managment issues: hiring, firing, internal controls, incident response plan,
  - Many attacks are inside jobs (see Thompson paper)
  - Many attacks are "social engineering" – ("Mr. Johnson, we are doing maintenance on your account. For verification purposes, what is your password?"; web site phishing; ...)
  - Technical person can't throw up your hands and say "not my problem" – (1) technical design decisions have usability implications (2) someone needs to handle interface between technical design and real-world use
- Technical issues – span computer science – not just crypto/authentication
  - Crypto (won't talk about much in this class)
    - Flaws in algorithm (break key) – main lesson here – Security by obscurity doesn't work. Best practice is to use public domain that have been widely reviewed. "Security lies in the key"
      See Bruce Schneier's "Snake Oil" discussion http://www.schneier.com/crypto-gram-9902.html
      "anyone can create a cryptography product that he himself cannot break. This means that a well-meaning person comes up with a new idea, or at least an idea that he has never heard of, cannot break it, and believes that he just discovered the magic elixir to cure all security problems. "
    - Interaction between low level crypto and higher-level protocols
      For the most part, crypto is a success story in modularity. Mortals don't need to understand details of a good crypto algorithm to use it properly
      .... But, as you build more sophisticated systems, need to understand details. (Example: JP Martin found a bug in a published "p2p backup" protocol that relied on secure hash for

    a challenge response protocol – I want to verify that you are storing my data by sending “random challenge” and you prove that you are storing my data by replying with MD5(data, “random challenge”); problem – can store partial computation on MD5(data) (which is much smaller than data) and then compute MD5 from partial state + challenge. Fix is simple: return MD5(“random challenge”, data) to prove you store data.
    Trend: more and more clever distributed protocols that rely on internal properties of crypto...

  - Authentication – discuss during next week or two
    * Tricky to get right, but good tools for doing so. Next week or so – Prudent practices, logic of authentication, SDSI, ...
  - Program bugs – dominant source of problems. Lots of good research issues here – verification, tracking causal paths, ...

    Need to worry both about inadvertant blunders that can be exploited by outsider and insider jobs.

    Consider problem of Navy wanting to deploy 10M-line black-box programs from vendors. How ensure that they don’t fail or leak information?
  - Design bugs – specification, use, ... (interface to “non-technical issues”)

- Tension between technical issues and operation – easy to come up with answers that make sense from a narrow technical point of view but that are unusable and may hurt secrurity in practice
  - Complex technical solutions can make system more difficult to use correctly
  - e.g., Password security – short passwords easy to guess; long obscure passwords people resist or write down (or, if we make logging in inconvenient enough, people will stay logged in all the time.)

3 Why Cryptosystems Fail

Ross Anderson

- plug: *Security Engineering* by Ross Anderson

  Lots of fun
  - Standard stuff like Chapter 2 Protocols, Chapter 3 Passwords, Chapter 4 access control; but also...
  - Chapter 11 - Nuclear command and control
  - P 19 - the MIG in the middle attack
  - P 267 Fingerprint identification in crimes ”Even if the probability of a false match .. is one in ten billion as claimed by police, once many prints are compared against each other, probability theory starts to bite. A system that worked well in the old days, whereby a crime scene print would be compared manually with the records of 57 known local burglars, breaks down once thousands of prints are compared every year with a database of millions.”
  - P 291-295 ”How to hack a smartcard (1-7)”
  - Chapter 17 - telecom security (phone phreaking and mobile phones)
  - ...Lots of fun...

  Lesson: A good computer security designer needs to be broad
  1. Learn from world around you
  2. Your systems must exist in the real world (payTV, road tolls, medical records, ... not just login to workstation)

- Threat model
  - Principle: Study why things really fail so you solve the right problem
  - Often: misplaced focus

    When the crypto know-how was originally imported from the defense sector, a threat model came with it. This model presumed that attacks on the system would be technically sophisticated, and might involve cryptanalysis or the manipulation of transactions at some point in the path between the ATM and the bank that issued the card.

    Designers...have suffered from a lack of feedback about how their products fail in practice, as opposed to how they might fail in theory. This has led to a false threat model being accepted; designers focused on what could possibly go wrong rather than what was likely to, and many of their products ended up being so complex and tricky to use, they caused implementation blunders that led to security failures."

    High-tech threats were the ones that most exercised the cryptographic equipment industry, and the ones that their products were explicitly designed to prevent. But these products are often so difficult to integrate into larger systems that they can contribute significantly to the blunders that caused the actual losses

    - In reality: "Of the hundreds of documented failures of ATM security, however, only two involved such attacks."
    - Conclusion: "Our research showed that the organizational problems of building and managing secure systems are so severe that they will frustrate any purely technical solution"
  - 3 main causes (of ATM fraud) ("Almost all attacks on banking systems involved blunders, insider involvement, or both.")
    - Program bugs: "Background noise" – difficult to get error rate below 1 in 10,000

      Lower bound on transaction error rate 1/10,000 due to program bugs for large, heterogeneous transaction processing system [note: cites a 1991 study here–has this number changed] → 600K ”phantom withdrawals” in US

      [Banks' response: phantom withdrawals clustered near residences of cards]
    - Postal service – PINs sent by mail
    - Theft by bank staff

      "British banks dismiss about 1% of their staff each year for disciplinary reasons..."
  - "More exotic attacks" – Variations on: Acquire PIN and card #
    - Basic programming errors (e.g., include PIN on card, "test mode" disgorges notes, fooled by telephone cards, ...)
    - Bad PIN design (e.g., 1+3 == 2+4 → correct)
    - Theft at interface (e.g., false ATM machine; or add card reader + small camera to real ATM machine, "shoulder surfing" + account number on receipt)
    - Possible help: Smart cards

      How do smart cards work?

      Are smart cards a magic bullet? (What problems do they solve? What problems don't they solve?)

      Why might banks (particularly in the US) be inclined to adopt smart cards? (Help prove customer did withdrawal. E.g., current mobile phone security put in place to enforce billing not customer privacy.)

- Generalization to other systems QUESTION: How do these issues manifest in other systems? Any other general issues?
    * Background noise – false alarms. If system often tells me “X wants to run Y and this might be insecure” or often tells me “this certificate isn’t quite right” and asks me to hit “OK”, I eventually get used to hitting “OK”. (Burgler alarm analogy)
    * Postal service – “If you’ve forgotten your password, press this button to have it emailed to you”
    * Theft by bank staff – insiders must be considered
    * Basic programming errors – yup
    * Bad PIN design – bad password design (easy to guess v. unrememberable)
    * Theft at interface – (kinda like phishing – email directing you to false web page scam; fake login page; ...)
    * Social engineering – “This is customer service, your password may have been compromised. Let me try to log in using your password...”
- Other data
    * “A recent US Air Force survey revealed that poor implementation is their main security problem”
    * “The National Security Agency has also recently admitted that most security failures in its area of interest occur at the level of implementation detail.”

Design principles/lessons

- Big picture
  - Don’t lose sight of the big picture and focus just on sophisticated technical means of attack; know why real systems get compromised and recognize that a system that is more complicated to design or use may be significantly less secure in practice than a simpler system with ”weaker” technological safeguards
  - ”Moral hazard” – The entity responsible for verifying authentication should be the one that pays the penalty for authentication errors.

    Very general principle - Traditional business practice for handwritten signatures ”In general, if someone wishes to enforce a document against you on the basis that you signed it, and you deny that you signed it, then it is for them to prove that the signature was made by you or authorized by you.” [_Security Engineering_ p 483]

    → balance convenience v. security [continuum: handshake deal, ”press 9 to agree”, faxed signature, original signature, compare original signature against reference [e.g., on back of card], witnessed signature, notarized signature, bank signature guarantee]

    “If...the system operator carries the risk, as in the United States [for ATM transactions], then the public-interest issue disappears, and security becomes a straightforward engineering problem for the bank (and its insurers and equipment suppliers).”

    Note: Legal/policy issue here - natural for entity to try to transfer the legal risk to the other party
    * Efforts to create a technical digital signature standard s.t. signatures are presumptively genuine
    * Phone model v. credit card model - you are generally liable for all charges made to your phone (GSM security is primarily there to prove that a call made by a customer not to protect your account from having minutes stolen...)

    * ...

  QUESTION: How does incorrect moral hazard definition in Britain contribute to three main causes of ATM phantom withdrawals.

- Expertise needed
  - "Equipment vendors may argue that skill in cryptology is rare...however, because [purchasers] lack skills at security integration and management, they will go on to build systems with 'holes."'
- Research challenge:
  - "How can we cope with the reality of unskilled implementers and multiple threats? One line of attack is to try to make security systems robust. This might mean that they can tolerate minor errors in system design, implementation, and operation with no loss of security, or with – at worst – a graceful degredation. Alternatively, we might try to design the components so they cannot be fitted together in unsafe ways. Robustness may be the most important problem in security, yet there has been very little research done on this topic."
- Design principle: Explicitness
  - Make security goals explicit
  - Protocols: Message meaning should be explicit in message content rather than inferred by, say, message sequence (see Abadi and Needham and Burrows, Abadi, Needham)
  - Formal methods (Example "A logic of Authentication") o Explicitly check for claimed properties o Most effective for compact subsystems (e.g., authentication protocols [BAN])
  - Integrate security and SW engineering (Example "Prudent engineering practices")

    o Subproblem: Identify which objects in system have security significance (requires explicit tracking of dependencies; Good compiler research problem?)

    o Explicit assumptions about possible interactions between modules
  - Adapt techniques from "Safety critical systems"

    1) "The specification should list all possible failure modes of the system"

    "The specification should list all possible failure modes of the system. This should include every substantially new accident or incident that has ever been reported and that is relevant to the equipment being specified."

    2) Explain what strategy has been adopted to prevent each of these failure modes

    3) Describe how each of strategy implemented – both technical design and related procedures, training – ensure that assumed skills are in place

    "[the specification] should then spell out how each of these strategies is implemented, including the consequences when each single component fails. This explanation must cover not only technical factors, but training and management issues too. If the procedure when an engine fails is to continue flying with the other engine, then what skills does a pilot need to do this and what are the procedures whereby these skills are acquired, kept current, and tested?"

    4) Review by independent experts; monitoring and incident reporting

  “The certification program must include a review by independent experts, and test whether the equipment can in fact be operated by people with the stated level of skill and experience. It must also include a monitoring program whereby all incidents are reported to both the equipment manufacturer and certification body.”

- Design principle: Robustness
  - Graceful degradation
  - Process for establishing (explicit) security goals – “A plain-language ‘concept of operations’ should be agreed upon before any detailed specification work is undertaken.”

    Example: “ATM Security involves several conflicting goals, including controlling internal and external fraud, and arbitrating disputes fairly. This was not understood in the 1970s; people built systems with the security technology they had available, rather than from any clear idea of what they were trying to do. In some countries, they ignored the need for arbitration altogether.”

    Example: Current controversy about e-voting technology. Many/most computer scientists argue for a voter-verifiable paper trail to allow recounts. Many/most vendors argue that this is not needed.

4 Reflections on trusting trust

A cautionary tale

- Moral of the story
  - Robustness?: No. Once someone has “root”, system can be completely subverted and practically all evidence removed. If your machine is hacked, throw away the disk drive!
  - Insider job (but fragility applies to any penetration)
  - Broad lesson: if superuser is evil, we’re all in trouble
    - ∗ Problem magnified in Unix by requiring lots of programs to have superuser privlege (more program → more opportunities for insider, more exploitable bugs, more misconfigurations)

      o Backup – needs to read all users files o Mail – need to copy data from protected shared mail file/socket to protected per-user mail file (→ sendmail follies) o ...
    - ∗ Possible solutions? Fine grained access control. Audit. Legal/technical constraints.

- The story
  - Step 1: modify login.c

```
if (password == "kensspecialpassword")
           return SUCCESS
```

  - Step 2: Hide change so no one can see it - modify C compiler

```
B:
if see trigger,
insert A into input stream
```

Whenever the compiler sees a trigger /* gobbleygook */, puts A into input stream of the compiler

Now, don't need A in login.c, just need the trigger

Need to get rid of problem in the compiler

- Step 3: Hide change to compiler (and password checker) by embedding it in compiler binaries. Modify compiler to have:

```
C:
if see trigger2
insert B + C into input stream
```

- Step 4: compile compiler with C present

  now in binary for compiler

- Step 5: replace code with trigger2

  Result is - all this stuff is only in binary for compiler. Inside the binary there is C; inside that code for B, inside that code for A. But source only needs trigger2

  Every time you recompile login.c, compiler inserts backdoor. Every time you recompile compiler, compiler re-inserts backdoor

  What happens when you port to a new machine? Need a compiler to generate new code; where does compiler run?

  On old machine - C compiler is written in C! So every time you go to a new machine, you infect the new compiler with the old one.

5 Environment

Computer designers design to make sure that software interfaces are secure. But software runs on hardware in the real world... (lessons – think about specifications; think about how system interacts with world; ...)

Tenex - early '70s BBN

Most popular systems at universities before Unix

Thought to be v. secure. To demonstrate it, created a team to try to find loopholes. Give them all source code and documentation (want code to be publicly available as in Unix). Give them a normal account

in 48 hours, had every password in the system

Here's the code for the password check in the kernel:

```
for(I = 0; I < 8; I++){
if(userPasswd[I] != realPasswd[I]
go to error
```

Looks innocuous - have to try all combinations - 256^8

But! Tenex also had virtual memory and it interacts badly with above code

Key idea - force page fault at carefully designed times to reveal password

Arrange first character in string to be last character in page, rest on next page. Arrange that the page with first character in memor, and rest on disk

```
a|aaaaaa
```

Time how long password check takes

- if fast - first character is wrong
- if slow - first character is right; page fault; one of others was wrong

so try all first characters until one is slow Then put first two characters in memory, rest on disk try all second characters until one is slow...

→ takes 256 * 8 to crack password

Fix is easy - don't stop until you look at all characters But how do you figure that out inadvance?

Timing bugs are REALLY hard to avoid!!

- smart card power supply analysis
- Tempest - your monitor (and keyboard) is also a radio transmitter - relatively easy to build a device that can receive radio broadcast and display what your monitor is displaying from several feet away (High end attack: irradiate the subject machine at resonance frequency of keyboard cable → pick up keystrokes from 50-100m. Some speculate this is why the USSR constantly beamed radar at the US embassy in Moscow for a while...)
- Traffic analysis - e.g., you encrypt your web traffic over network so know one knows what you are browsing. But they see 14321 bytes, pause, 29140 bytes, pause, 2341 bytes, pause...Pretty quickly they can match what pages you are viewing to a suspect website with high confidence
- ...

6 Crypto basics

TBD: Insert basics of authentication + encryption from 372 27.doc here

7 Prudent Engineering Practices: Excerpts and commentary

Abadi and Needham paper

7.1 Abstract

We present principles for designing cryptographic protocols. The principles are neither necessary nor sufficient for correctness. They are however helpful, in that adherence to them would have prevented a number of published errors.

Our principles are informal guidelines; they compliment formal methods, but do not assume them....

We arrived at our principles by noticing some common features among protocols that are difficult to analyze. If these features are avoided, it becomes less necessary to resort to formal tools—and easier to do so if there is good reason to."

7.2 1.

"Every message should say what it means: the interpretation of the message should only depend on its content."

"...If any of P, S, A, B, or K are left to be inferred from context, it may be possible for one message to be used deceitfully in place of another..."

A common technique: include hash of all preceeding messages in each message (e.g., Zhou's COCA protocol)."

Advice:

- When analyzing a protocol, blank out the "A $\rightarrow$ B" part or anything not encrypted. Just imagine that the series of messages are broadcast by a party unknown and consider what each node is entitled to believe after each broadcast. (Think of this as proving safety; to prove liveness, you need to ensure that the intended party has enough information to issue a message at each step. Unencrypted stuff can be treated as hints, but the safety of the protocol cannot depend on them.)
  - *can* you build a protocol that depends on message order or plain-text for correctness? Sure. But it is not *prudent* to do so because doing so *tends* to be error-prone.
- A related point: if two parts of a message are not "bound together" by encryption, pretend that the protocol actually sends them in two separate messages.
- Abadi and Needham suggest writing the "english" intepretation of the message; any quanity or entity mentioned in the prose had better appear in the message.

7.3 2.

"The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not."

"...these conditions often consist of what may be regarded informally as statements of trust...Statements of trust cannot be wrong, although they may be considered inappropriate. For example, if someone believes that choosing sessions keys should be done by a suitably trusted server rather than by one of the participants in a session..."

E.g., Logic of authentication requires you to list assumptions explicitly (vy useful.)

7.4 3.

"If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principals name explicitly in the message."

[in the Denning and Sacco protocol] "Finally, B should know that the message was intended for B (because of the use of KB). Unfortunately, nothing provides this final guarantee, with dramatic consequences. Any principal B with which A opens communication can pretend to a third party C that it actually is A." [a man in the middle attack]

1. A $\rightarrow$ S: A, B
2. S $\rightarrow$ A: CA, CB
3. A $\rightarrow$ B: $CA, CB, K_{ab}, {{T_a}_{K_a^{-1}}}_{K_b}$

Notice that by following the rule of thumb above "blank out anything not encrypted", this would be fairly obvious.

Look at just the encrypted message – the "real protocol"– (steps 1 and 2 just mean "(1) hint: start protocol for A, B and (2) precondition: everyone knows eveyrone's name and certificate"):

3 A $\rightarrow$ B: $K_{ab}, {{T_a}_{K_a^{-1}}}_{K_b}$

This is the only encrypted message B receives. What can B know after seeing it?

- Want: "**A says** that K_{ab} is **now** a good key for communication **between A and B**"
  - Encrypted by A's private key gives us "A says"

  - T_a gives us "now"
  - K_{ab} gives us K_{ab}
  - **What gives us "between A and B"?**
    - Maybe: the two keys K_a^-1 and K_b?
    - No: K_b is a public key...seeing things tied together with K_b lets B know that only it can read the message, but it doesn't say anything about who sent the message

- Actually get: "**A says** that K_{ab} is **now** a good key for communication (with **someone**)"
- Problem:

  3 A → B: $CA, CB, K_{ab}, T_{a_{K_a^{-1}}}{}_{K_b}$

  3' B → C: $CA, CC, K_{ab}, T_{a_{K_a^{-1}}}{}_{K_c}$

  - Result:
  - $\rightarrow C$ thinks K_{ab} is a good key for communicating with A (but B knows K_{ab}

- Solution: add "A, B" to the stuff signed by K_a^{-1}

[in the Lu and Sundareshan protocol] "the fundamental flaw of the protocol is rather simple. One immediately sees that neither A nor B ever receives a message that contains the others name...[allowing] some easy attacks to defeat the protocol."

7.5 4.

"Be clear about why encryption is being done. Encryption is not wholly cheap, and not asking precisely why it is being done can lead to redundancy. Encryption is not synonymous with security, and its improper use can lead to errors."

Opinion: Encryption is cheap. Don't be too concerned about optimizing performance. *Do* be concerned about clarity or purpose.

Uses of encryption include: confidentiality, authenticity, bind together parts of a message, and producing random numbers.

7.6 5.

"When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message. On the other hand, it is proper to assume that a principal that signs a message and then encrypts it for privacy knows the content of the message."

[e.g. the CCITT protocol] "this corresponds to a scenario where some third party intercepts a message and removes the existing signature while adding his own, blindly copying the encrypted section within the signed message."

Point is: this protocol (in example 5.1 and 5.2) doesn't actually guarantee that A originated the message.

Rule of thumb: sign then encrypt rather than encrypt then sign

7.7 6.

"Be clear what properties you are assuming about nounces. What may do for ensuring temporal succession may not do for ensuring association – and perhaps association is best established by other means."

["other means" put relevant state explicitly in messages as per principle 1 to get simpler and clearer protocols.]

7.8 7.

"The use of a predictable quantity (such as the value of a counter) can serve in guaranteeing newness, through a challenge-response exchange. But if a predictable quantity is to be effective, it should be protected so that an intruder cannot simulate a challenge and then later replay a response."

[Predictable nounces must generally be encrypted. Not for secrecy. Encryption serves to construct a quantity that only A can predict from a quantity anyone can predict.]

7.9 8.

"If timestamps are used as freshness guarantees by reference to absolute time, then the difference between local clocks at various machines must be much less than the allowable age of a message deemed to be valid. Furthermore, the time maintenance mechanism everywhere becomes part of the trused computing base."

[slow clock –¿ may mistake expired certificates for current ones. Fast clock allows replay attacks.]

7.10 9.

"A key may have been used recently, for example to encrypt a nonce, yet be quite old, and possibly compromised. Recent use does not make the key look any better than it would otherwise."

7.11 10.

"If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used."

7.12 11.

"The protocol designer should know which trust relations in his protocol depends on, and why the dependence is necessary. The reasons for particular trust relations being acceptable should be explicit, though they will be founded on judgement and policy rather than on logic."

definition *trust* – A trusts B in regard to some function if a loss of security to A could follow from B not behaving in some specified way

This related back to Anderson's "robustness" ideas – know what part of your system are critical to security; know what will happen if they fail. (E.g., Abadi and Needham: "Complete loss of security could follow from a Kerberos server issuing wrong timestamps")

7.13 Conclusion

We have found the principles and examples described in this paper useful in our own work. Perhaps it is because of this that they bear a certain subjective character. We do however believe that they respond to an important general need in a discipline where some basic mistakes appear in print several times.

Many of our suggestions can be embodied in development methods and in formalisms. While these are helpful, we tried to emphasize an informal understanding of some issues essential for security. We hope that our guidelines will improve the practice of designing cryptographic protocols.

8 Admin

9 Saltzer and Schroeder

“Proection of information in computer systems

9.1 Principles

1. economy of mechanism (keep design simple)
2. Fail-safe defaults
3. Complete mediation
4. Open design
5. Separation of privlege
6. Lease privlege – every program and user should operate using the least set of privleges needed to do the job
7. Least common mechanism – minimize mechanism depended on by all users
8. Psynchological acceptability