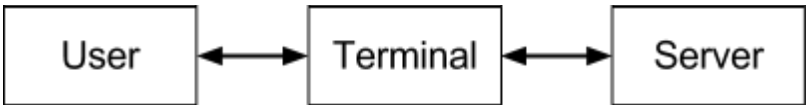
(Broad) secure transaction problem



The terminal may be untrustworthy. In case of ATM, it is probably not a primary threat, but it might have been modified (for example, card skimmer, as we addressed in a previous lecture). For online banking, user's laptop may have some malware on it.

How can we make secure transactions, assuming a not-so-trustworthy terminal? There are two different approaches people have been working on.

1. Making terminal trustworthy as much as possible.
    a. Live CD
    b. Separate PC for mission-critical usages, such as financial transactions, while using primary PC for ordinary use, such as web browsing (often recommended for small businesses)
2. Secondary device for secure authentication (security id/token/card)
    a. Small hardware with limited purpose (no arbitrary code on it), optionally with I/O such as LED and keypad, with some combination of
        i. Random number generator
        ii. Asymmetric/symmetric cryptography
        iii. Counter
        iv. Synchronized clock (GPS?)
    b. Usage example
        i. When you want to conduct a transaction, use your terminal as usual
        ii. Just before complete the transaction, the server asks 'give me the code on your token'
        iii. User presses the button on the token. It shows 6-digit confirmation code. User types the code.
        iv. The backend server and the token share the same cryptographic key, so the server can verify the code.

The secure token does do well depending against key loggers or over-the-shoulder attacks; no authentication credentials can be leaked. One downside is man-in-the-middle (MITM) attack. Suppose your laptop is infected by malware. The malware is effectively the man in the middle, so it can arbitrarily change my transaction. For instance, it can replace "withdraw $100 from my checking account" with "transfer $1,000 from my account to Malice's account." What can we do about this?

One potential solution could be "typing into the token". Say, the token has a card slot, a display, and a keypad, you type all the transaction information (amount, recipient, type of transaction, etc.) into the

token. There are some issues with it: cost, usability issues (battery, user experience, carrying the token always, etc.)

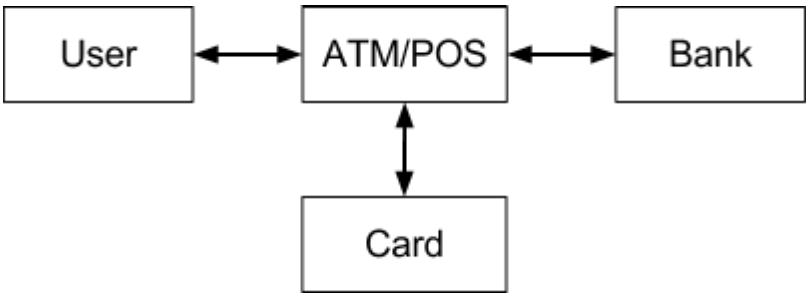
> Suggestion: Why bother with secure token? Everyone has a smartphone, and we carry it everyday. We can use our smartphones as a terminal + secure token.
>
> Discussion: Your smartphone is basically a general-purpose computer. Lots of activities (e.g., web browsing) happens on the smartphone, and it can run arbitrary code on it. So it is not very trustworthy. Even worse, if you have a PC to sync data with the smartphone and the PC is compromised by sophisticated malware, it might be able to infect your phone as well. The advantage of secure token comes from its limited capability (so it cannot go wrong and you can make it very cheap).

The conventional wisdom says if your token does not have both display or input capability, then it is not going to be secure against malware because of man-in-the-middle attack. In this case, the user must interact with the terminal so the user does not have any defense against man-in-the-middle attack, since the secure token does not tell you what is exactly going on the transaction.

Another issue: what happens if the token is stolen?

- The attacker can perform authentication to impersonate you.
- Standard solution: 2-factor authentication
    - Token (physical item you have)
    - Password (some knowledge you know)
- So even if the token is stolen, he cannot proceed without your password.

Chip and Pin



- Terminal: a ATM machine or a Point-Of-Sale terminal
- Backend server: the bank (issuer of the card)
- Secure token: credit/debit card (with a chip on it)
    - This is what you have.
    - It does not have any user interface because of cost issues. No input/display
    - So it directly interacts with the terminal, not the user.
- PIN: what you know.

The pin you type goes to the card, then the card can verify whether it is correct or not. Then some cryptographic authentication with a challenge/response protocol happens. The card performs signing on the transaction details with message authentication code (MAC). The correctness of the MAC is confirmed by the central bank server, which has the same cryptographic key.

Note that chip and pin does not solve the MITM vulnerability issue. Suppose that a corrupt merchant tampers with his POS. Then he can manipulate transactions: for instance, charging $1,000 for a candy bar, showing only $1 on display to the customer. While this could be problematic, the risk is more or less acceptable; this attack is fairly detectable, legal consequence is severe, clerks do not have clear motivation, etc.

What about stolen cards?

- The attacker does not know its PIN. What can (s)he do with it?
- One attack model presented in previous work
    - MITM between the terminal and the (stolen card)
    - Step 1: The crook enters an incorrect PIN.
    - Step 2: The MITM device (tiny attachment to the card) hijacks the PIN verification message from the POS and says it is correct.
        - one variation: the MITM device fools the card pretending that it is signature mode, not PIN mode.
    - Step 3: PROFIT! The crook can buy stuff with a stolen card not knowing its PIN.
- Pre-play attack discussed in the paper, outline:
    - The transaction includes a nonce, which is a unpredictable number. This is for freshness.
    - Some ATM happen to have a very poor, predictable random number generator (RNG).

    - Step 1: By using a corrupt terminal, challenge the card with a likely-to-be-used nonce. Record the response from the card.
    - Step 2: Replay the response on a legitimate ATM, if the predicted nonce is challenged.

QUIZ: One possible solution would be letting the bank generate a nonce, not the terminal. But it requires an additional round-trip latency. How can we achieve similar security without an additional RTT? (hint: distributed random number generation)

Lesson from the pre-play attack (difficulty of multi-party protocol design)

- Who is responsible and who has incentives are often different.
- This leads to sloppy implementation (e.g., poor RNG).

Other things discussed in the class

- Testing security systems is difficult.
    - For instance, how to verify the quality of RNG?
    - Suppose that an algorithm has very good statistical randomness, but what if the seed is predictable?
- The pre-play paper mentions a small clone-mitigation mechanism
    - 16-bit counter in the chip. What is good for?
    - Why is it not enough?
    - The concept of Hash Chaining for better security
        - So the bank can check if the hash values diverge due to cloning.