Basics of Cryptography, Cryptoprotocols, and Steganography

14 August 2017

Clark Thomborson

University of Auckland

Security Requirements

- Alice wants to send a message to Bob. Moreover, Alice wants to send the message securely: Alice wants to make sure Eve cannot read the message."

  – [Adapted from Schneier, *Applied Cryptography*, 2nd edition, 1996]

- Exercise 1. Draw a picture of this scenario.

- Exercise 2. Discuss Alice's security requirements, using the terminology developed to date in CompSci 725.

- Exercise 3. In this scenario, Alice is the sender, Bob is the receiver, and Eve is the eavesdropper. Name another actor with an important role in communication security.

  – Sample answers are widely available on the internet, see e.g. http://en.wikipedia.org/wiki/Alice_and_Bob.

ALICE AND BOB

Alice

Bob

Information source

Information destination

(a) Normal flow

Alice

Bob

Information source

Information destination

(b) Interruption

Alice

Bob

Information source

Oscar

Information destination

(c) Interception

Alice

Bob

Information source

Oscar

Information destination

(d) Modification

Alice

Bob

Information source

Oscar

Information destination

(e) Fabrication

Alice

Bob

Information source

detect

Eve

Information destination

(f) Stegocommunication

Fig. 1.   Normal message flow, and five fundamental threats to this flow.

14-Aug-17

From "A Security Model for VoIP Steganography", by Yu, Thomborson et al., DOI 10.1109/MINES.2009.227, 2009.

4

An Attack Taxonomy for Communication Systems

1. *Interception* (attacker reads the message)
2. *Interruption* (attacker prevents delivery)
3. *Modification* (attacker changes the message)
4. *Fabrication* (attacker injects a message)
   a) *Impersonation* (attacker pretends to be a legitimate sender or receiver, e.g. this is either a fabrication or an interruption)
5. *Stegocommunication* (Alice and Bob make surreptitious use of a communication system; Eve wears a "white hat")
6. *Repudiation* (a black-hat Alice falsely asserts she did not send a message to Bob, or a black-hat Bob falsely asserts that he didn't receive a message from Alice); white-hat Judy is the judge.

Symmetric and Public-Key Encryption

- If the decryption key `d` can be computed from the encryption key `e`, then the algorithm is called *symmetric*.
  - Example: `E(p) = (p + e) mod 256` is a symmetric (and very weak) encryption of a char `p`, because `D(x) = (x + d) mod 256` is a decryptor when `d = 256 - e`.

- If the decryption key cannot be feasibly computed from the encryption key, then the algorithm is called *asymmetric* or *public-key*.

Message Integrity

- Encryption assures confidentiality
  - Assume: the attacker can't discover the key or "crack" the cypher.
- Integrity can also be assured by message codes.
- Sending a plaintext message, plus its Message Authentication Code (MAC), will ensure message integrity to anyone who knows the (shared) secret key.
  - The CBC-MAC is the last ciphertext block from a CBC-mode block cipher.
  - Changing any message bit will change the MAC – this defends against modification.
  - Unless you know the secret key, you can't compute a MAC from the plaintext – this defends against fabrication.
- Keyed hashes (HMACs) are another popular type of MAC.
  - SHA-1 and MD5 are used in SSL
  - To learn more, read Stamp's *Information Security*, 2nd Edition, Wiley, 2011, at pp. 136-7.

MAC
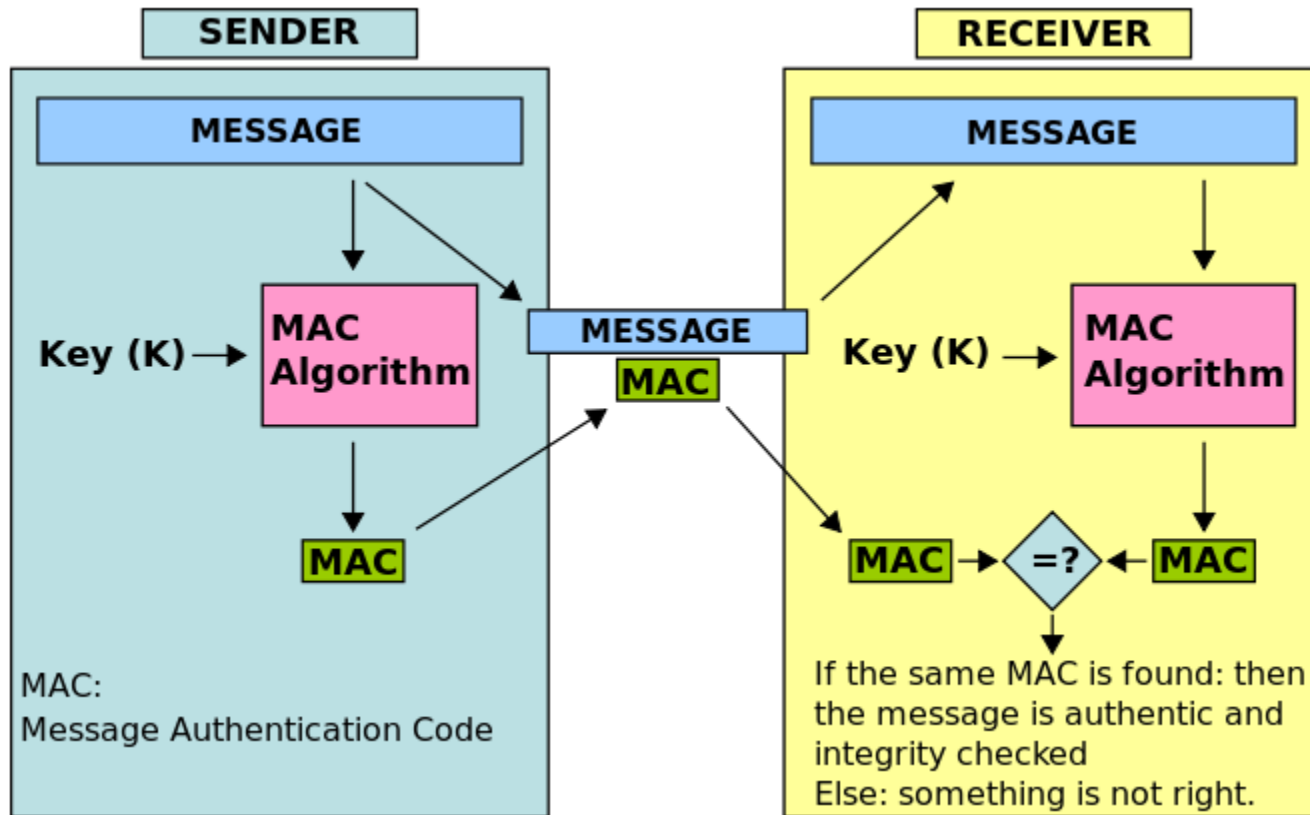
http://en.wikipedia.org/wiki/Message_authentication_code

Public Key Cryptography

Encryption *E: Plaintext* × *EncryptionKey* → *Cyphertext*

Decryption *D: Cyphertext* × *DecryptionKey* → *Plaintext*

- The sender must know the encryption key.
- The receiver can decrypt, if they know the decryption key.
- In *public-key cryptography*, we use key-pairs (*s*, *p*), where our secret key *s* cannot be computed efficiently (as far as anyone knows) from our public key *p* and our encrypted messages.
  - The algorithms (*E*, *D*) are standardized.
  - We let everyone know our public key *p*.
  - We don't let anyone else know our corresponding secret key *s*.
  - Anybody can send us encrypted messages using *E*(*, p*).
  - Convenient notation: {*P*}$_{Alice}$ is plaintext *P* that has been encrypted by a secret key named "Alice". [Stamp, pp. 89-91, 323]

Authentication in PK Cryptography

- We can use a secret key *s* to encrypt a message which everyone can decrypt using our corresponding public key *p*.
  - $E(P, s)$ is a "signed message". Simpler notation: $[P]_{\textbf{Alice}}$
  - Only people who know the secret key named "Alice" can create this signature.
  - Anyone who knows the public key for "Alice" can validate this signature.
  - This defends against impersonation and repudiation attacks.

- If you use a key-pair (s, p) for encryption, then you can't use it safely for signing!
 - Do you understand why?

Key Management & Distribution

- We should use many different public/private key pairs:
  - For our email,
  - For our bank account (our partner knows this private key too),
  - For our workgroup (shared with other members), …
- A "public key infrastructure" (PKI) will help us create, publicise, and discover public keys (p_1, p_2, …).
- A "certificate authority" (CA) is a registry for public keys – this is an important part of a PKI..
  - The CA uses one of its signing keys to sign a "certificate" of the form [*name, p*]$_{CA}$.
  - Anyone who knows the CA's corresponding public key can verify that *p* was registered by someone who convinced the CA that they are identified by *name*.
  - Note: we also need some way to discover CAs and their keys… our web browsers help with this…

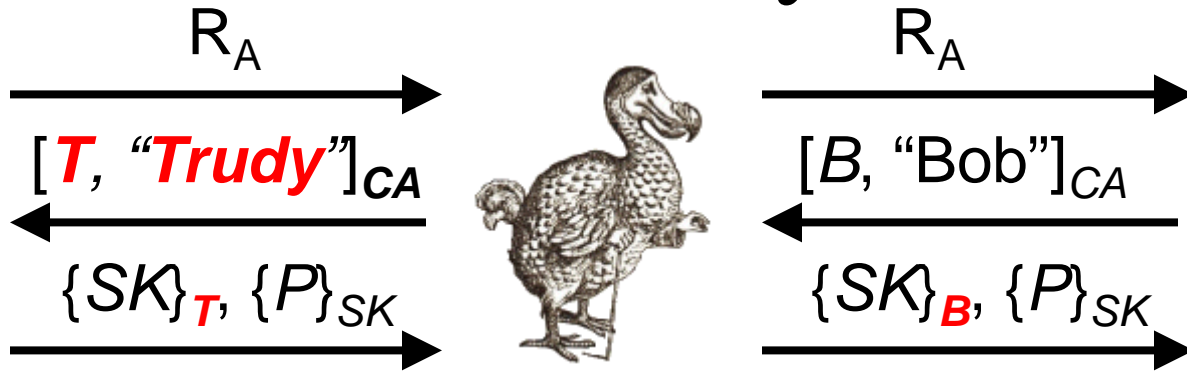# Some Security Issues with CAs

- The *name* in a certificate might not be a unique identifier for a person or an organisation – there are many people named "John Doe".

- A CA might register a key to an impersonator.

- The end-user might not inspect the certificate to confirm that

  – *name* is a (reasonably) unique identifier for the person or organisation they are trying to communicate with.

A Simple Cryptographic Protocol

$$R_A$$

$$[B, \text{"Bob"}]_{CA}$$

$$\{SK\}_B, \{P\}_{SK}$$
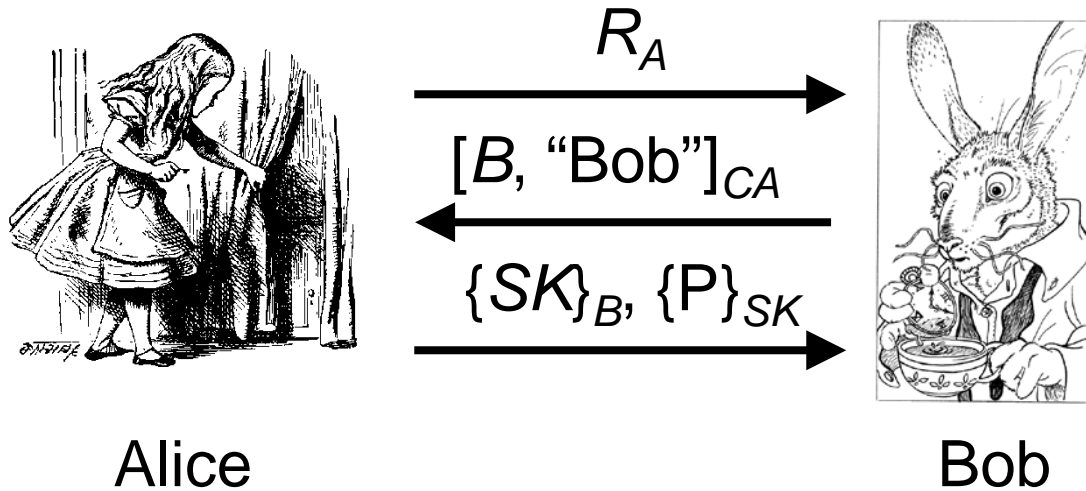
Alice   Bob

1. Alice sends a service request R_A to Bob.

2. Bob replies with his digital certificate.

   - Bob's certificate contains Bob's public key B and Bob's name.

   - This certificate was signed by a Certificate Authority, using a public key CA which Alice already knows.

3. Alice creates a symmetric key SK. This is a "session key".

   - Alice sends SK to Bob, encrypted with public key B.

   - Alice and Bob will use SK to encrypt their plaintext messages.

Protocol Analysis

$$R_A \longrightarrow$$

$$[\textbf{\textit{T}}, \textbf{\textit{"Trudy"}}]_{CA} \longleftarrow$$

$$\{SK\}_{\textbf{\textit{T}}}, \{P\}_{SK} \longrightarrow$$

$$R_A \longrightarrow$$

$$[B, \text{"Bob"}]_{CA} \longleftarrow$$

$$\{SK\}_{\textbf{\textit{B}}}, \{P\}_{SK} \longrightarrow$$

Trudy: acting as Alice to Bob,
and as Bob to Alice

Alice

Bob

- How can Alice detect that Trudy is "in the middle"?
- What does your web-browser do, when it receives a digital certificate that says "Trudy" instead of "Bob"?
- Trudy's certificate might be [T, "Bob"]$_{CA}$,
  - If you follow a URL to "https://www.bankofamerica.org", your browser might form an SSL connection with a Nigerian website which spoofs the website of a legitimate bank, or a <u>website controlled by a disgruntled BoA customer</u>.
- Have you ever inspected an SSL certificate?

Attacks on Cryptographic Protocols

- A ciphertext may be broken by…
  - Discovering the "restricted" algorithm (if the algorithm doesn't require a key).
  - Discovering the key by non-cryptographic means (bribery, theft, 'just asking').
  - Discovering the key by "brute-force search" (through all possible keys).
  - Discovering the key by cryptanalysis based on other information, such as known pairs of (plaintext, ciphertext).

- The weakest point in the system may not be its cryptography!
  - See Ferguson & Schneier, *Practical Cryptography,* 2003.
  - For example: you should consider what identification was required, when a CA accepted a key, before you accept any public key from that CA as a "proof of identity".

Limitations and Usage of PKI

- If a Certificate Authority is offline, or if you can't be bothered to wait for a response, you will use the public keys stored in your local computer.
  - Warning: a public key may be revoked at any time, e.g. if someone reports their key was stolen.

- Key Continuity Management is an alternative to CAs.
  - The first time someone presents a key, *you* decide whether or not to accept it.
  - When someone presents a key that you have accepted previously, it's ok to accept it again if you haven't had any bad experiences with that key,
  - If someone presents a changed key, you should think carefully before accepting!
  - This idea was introduced in SSH, in 1996. It was named, and identified as a general design principle, by Peter Gutmann (http://www.cs.auckland.ac.nz/~pgut001/).
  - Reference: Simson Garfinkel, in http://www.simson.net/thesis/pki3.pdf

Identification and Authentication

- You can authenticate your identity to a local machine by
  - what you have (e.g. a smart card),
  - what you know (e.g. a password),
  - what you "are" (e.g. your thumbprint or handwriting)
- After you have authenticated yourself locally, then you can use cryptographic protocols to…
  - … authenticate your outgoing messages (if others know your public key);
  - … verify the integrity of your incoming messages (if you know your correspondents' public keys);
  - … send confidential messages to other people (if you know their public keys).
  - Warning: you (and others) must trust the operations of your local machine! We'll return to this subject…

Steganography

- The art of sending undetectable messages.
  - The primary goal of the wardens is detection of stegocommunication.
  - The primary goal of the prisoners is availability.
  - It's up to the analyst to decide the colours of the hats!
    - Steganography, like cryptography, may be used by black-hats or white-hats.
- Steganography is *complementary* to cryptography.
  - Using strong cryptography, Alice and Bob achieve confidentiality and integrity.
  - Alice and Bob should use steganography if they're worried about availability or traffic analysis.
    - Cryptographic communications are "obviously" encrypted.
  - If warden Walter can't understand what Alice is saying…
    - Should he punish Alice for sending an encrypted message?
    - Should he prevent Alice's encrypted message from reaching Bob?
    - Should he carefully watch Bob, after allowing him to read the message?

Wardens and Prisoners

- "On July 17 [1965], a prisoner [in Mt Eden Prison] asked a guard to pass a newspaper to another prisoner in another cell.
- "The guard found a coded note in its pages.
  - Unable to decipher the message he simply copied it for the file.
- "Inexplicably, he then delivered the newspaper and its mysterious contents.
  - If that note had been successfully read, what occurred next would have been avoided.
  - … The prisoners began smashing up the central office and set it on fire at the same time other prisoners were being unlocked.
  - What the *Herald* would later call a 'wild orgy of destruction' ensured firefighters entering the jail were forced to retreat. …"

["The night all hell broke loose at Mt Eden Prison", NZ Herald, 28 July 2015]

Watermarking, Tamper-Proofing and Obfuscation – Tools for Software Protection

Christian Collberg & Clark Thomborson

IEEE Transactions on Software Engineering 28:8, 735-746, August 2002.

Watermarking and Fingerprinting

Watermark: an additional message, embedded into a cover message.

Copyright 1990 C. Collberg

Customer #314

- Messages may be images, audio, video, text, executables, …
- Visible or invisible (steganographic) embeddings
- Robust (difficult to remove) or fragile (guaranteed to be removed) if cover is distorted.
- Watermarking (only one extra message per cover) or fingerprinting (different versions of the cover carry different messages).

Our Desiderata for (Robust, Invisible) SW Watermarks

- Watermarks should be stealthy -- difficult for an adversary to locate.

- Watermarks should be resilient to attack -- resisting attempts at removal even if they are located.

- Watermarks should have a high data-rate -- so that we can store a meaningful message without significantly increasing the size of the object.

Attacks on Watermarks

- **Subtractive** attacks: remove the watermark (WM) without damaging the cover.

- **Additive** attacks: add a new WM without revealing "which WM was added first".

- **Distortive** attacks: modify the WM without damaging the cover.

- **Collusive** attacks: examine two fingerprinted objects, or a watermarked object and its unwatermarked cover; find the differences; construct a new object without a recognisable mark.

Defenses for Robust Software Watermarks

- **Obfuscation**: we can modify the software, so that a reverse engineer will have great difficulty figuring out how to reproduce the cover without also reproducing the WM.

- **Tamperproofing**: we can add integrity-checking code that (almost always) renders it unusable if the object is modified.

Classification of Software Watermarks

- Static code watermarks are stored in the section of the executable that contains instructions.

- Static data watermarks are stored in other sections of the executable.

☞ Dynamic data watermarks are stored in a program's execution state. Such watermarks are resilient to distortive (obfuscation) attacks.

Dynamic Watermarks

- **Easter Eggs** are revealed to any end-user who types a special input sequence.

- **Execution Trace Watermarks** are carried (steganographically) in the instruction execution sequence of a program, when it is given a special input.

☞**Data Structure Watermarks** are built (steganographically) by a program, when it is given a special input sequence (possibly null).

Easter Eggs
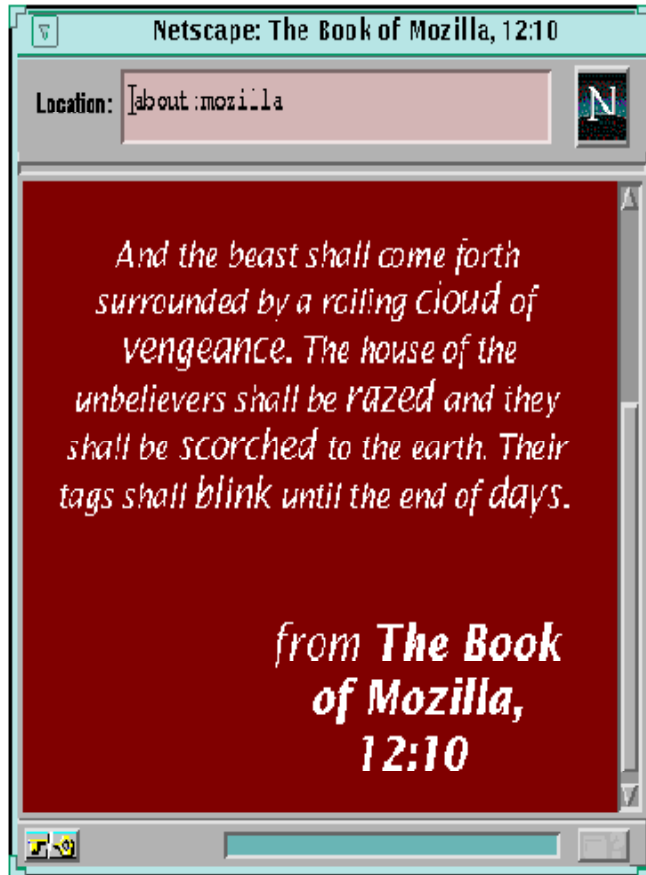
- The watermark is visible -- if you know where to look!

- Not resilient, once the secret is out.

- See www.eeggs.com

Software Obfuscation

- Many authors, websites and even a few commercial products offer "automatic obfuscation" as a defense against reverse engineering.

- Existing products generally operate at the lexical level of software, for example by removing or scrambling the names of identifiers.

- We were the first (in 1997) to use "opaque predicates" to obfuscate the control structure of software.

Opaque Predicates

$\{A; B\} \Rightarrow$

"always true" "indeterminate" "tamperproof"
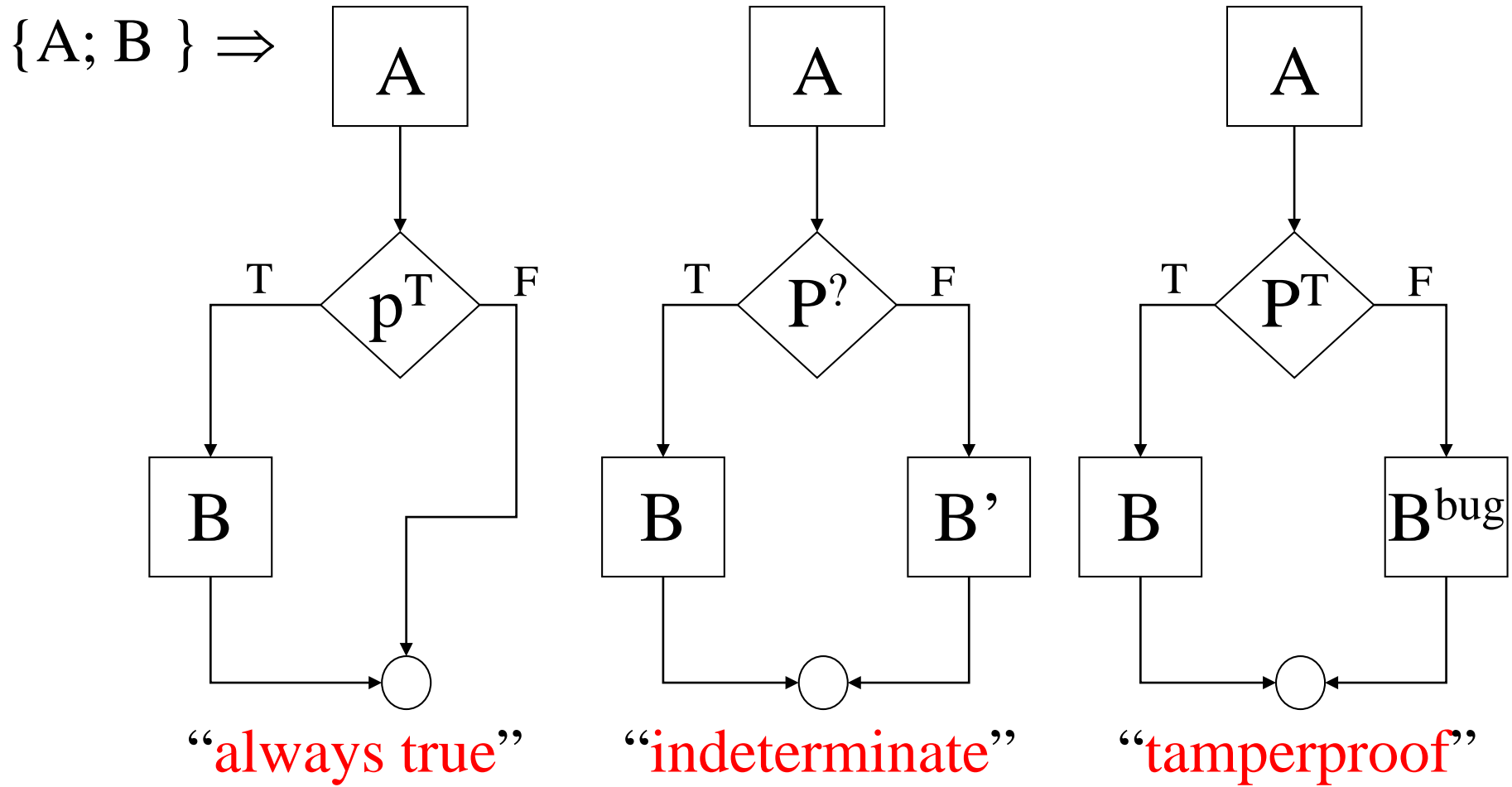
Conclusion

- Software obfuscation can make it more difficult for pirates to defeat standard tamperproofing mechanisms, or to engage in other forms of reverse engineering.

- Software watermarking can embed "ownership marks" in software, making it difficult for anyone to be sure that they have "removed all the marks".

- Software steganography is immature:
  – More R&D is required before robust obfuscating and watermarking tools will be easy to use.