Reminder:

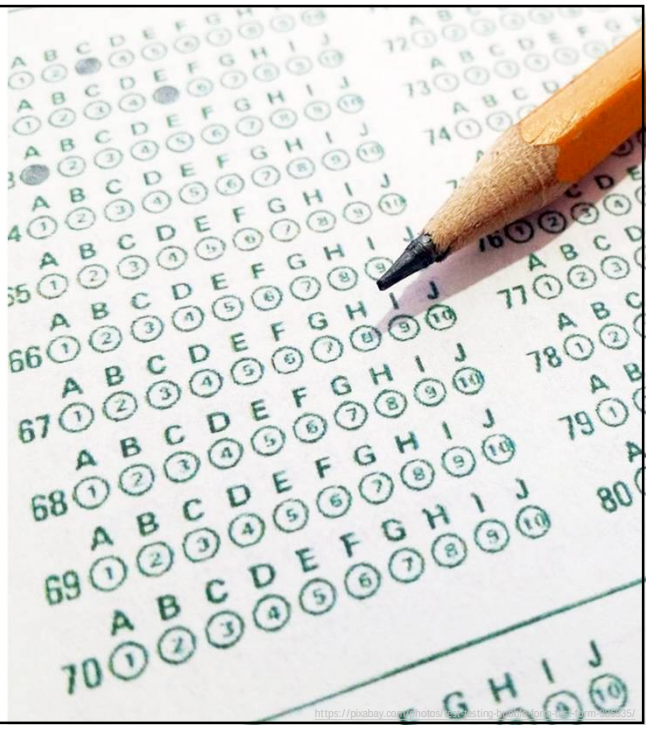
Please complete all "student evaluations of instruction."

There are several such evaluations being sent out from different offices of Dordt University.

I hear that some professors are requesting students to inform them when the evaluations have been completed. I prefer not to ask that of you.

That said, in a class as small as ours 100% participation from all students in all requested evaluations is important. Even one student not participating will seriously reduce the significance of the results.

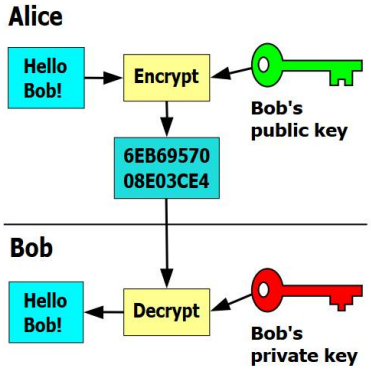
Please participate in all student evaluations of instruction.



1

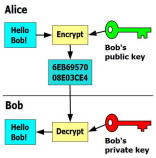
Public-Key Cryptography—a.k.a. Asymmetric Cryptography

Each person has two keys, a public key and a private key.
The sender requests the recipient's public key using an insecure channel.
The recipient sends the key using an insecure channel. We assume the key has arrived uncorrupted.
(Some arrangement needs to be made to authenticate the recipient to the sender.
Otherwise an enemy could send their public key and replace the intended recipient in the exchange.)
The public key of the recipient is used for encryption.
The private key of the recipient is used for decryption.



2

Public-Key Cryptography—a.k.a. Asymmetric Cryptography



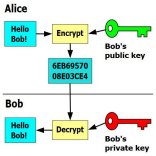
This system relies on two functions that are fundamentally different, yet the one inverts the other.
Let M represent the plaintext. (M may be for example a 128-bit integer, i.e. 16 bytes of data.)
Let C $= F(M)$ represent the cyphertext, where in essence, the function F is the public key.
Let $M' = G(C)$ represent the decrypted plaintext (if successful, $M' = M$), where in essence G is the private key.

The function F needs to be a "one-way" function such that M cannot be recovered from C.
The function G needs to be the inverse of F.
The key G must not be able to be derived from F.
Amazingly, this is possible!

https://commons.wikimedia.org/wiki/File:Public_key_encryption.svg

3

Public-Key Cryptography—a.k.a. Asymmetric Cryptography



This system relies on two functions that are fundamentally different, yet the one inverts the other.
Let M represent the plaintext. (M may be for example a 128-bit integer, i.e. 16 bytes of data.)
Let C $= F(M)$ represent the cyphertext, where in essence, the function F is the public key.
Let $M' = G(C)$ represent the decrypted plaintext (if successful, $M' = M$), where in essence G is the private key.

The function F needs to be a "one-way" function such that M cannot be recovered from C.The function G needs to be the inverse of F.
The key G must not be able to be derived from F.
Amazingly, this is possible!

An example—too simplistic to really work, but it gives the idea.

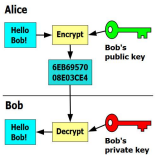
Suppose the message to be sent will always be an odometer reading from a passenger car.
It will always be between 00 and some prime number, say 96. (Should the car last long enough, it will roll over.)
Let's also assume that **subtraction is a ridiculously difficult task**—consider it practically impossible.

Key generation: Pick a pair of integers e and d such that $0 < e < 96$ and $e + d = 97$.
Public key is e, private key is d (neither e nor d will be factors of 97 because 97 is prime)
To encrypt: $C = (M + e) \bmod (n)$ Notice: without d, can't decrypt. Encryption is "one-way."
To decrypt: $M' = (C + d) \bmod (n) = \big((M + e) \bmod (n) + d\big) \bmod (n) = M$ (No subtraction needed.)

https://commons.wikimedia.org/wiki/File:Public_key_encryption.svg

4

RSA—a type of public-key cryptography



Named after Rivest, Shamir, and Adleman.
Operates on blocks of text, typically 128 bits of text at a time.
Every 128-bit long string of plaintext gets replaced with another 128-bit long string of cyphertext.
(and vice versa to decrypt.)

Key generation:
Start with two large prime numbers, p and q
Multiply them together. $n = pq$
Also compute Euler's Totient Function, which in this case is $\phi(n) = (p-1)(q-1)$
Find an integer e such that $1 < e < \phi(n)$ and there is no common divisor of e and $\phi(n)$ (1 is not a GCD by def'n).
Find yet another integer d such that $1 < d < \phi(n)$ and such that $ed \bmod(\phi(n)) = 1$
The public key (typically used for encryption of plaintext) is the pair (n, e)
The private key (typically used for decryption of cryptotext) is the pair (n, d)

Encryption: $C = F(M) = M^e \bmod(n)$

Decryption: $M' = G(C) = C^d \bmod(n) = M^{ed} \bmod(n) = M^1 = M$

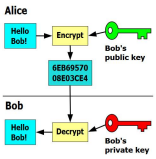
Overall: $M' = G(F(M))$ and $M' = M$

Interestingly, for most of these systems $G(F(M)) = F(G(M)$ ←important!
This means the roles of the keys may be interchanged. (If done consistently.)

https://commons.wikimedia.org/wiki/File:Public_key_encryption.svg

5

RSA—a type of public-key cryptography



The mathematical algorithms of RSA need to be sophisticated so that the computations do not take up too much computer time.

Especially on embedded platforms resources may be thin. Use a library!

https://commons.wikimedia.org/wiki/File:Public_key_encryption.svg

6

RSA—a type of public-key cryptography

Authentication of the source of the message is an issue with RSA. (Must prevent "person-in-the-middle attacks.)

Suppose Alice requests Bob's public key. Presumably, she must use an insecure channel
(Else use the secure channel and forget about cryptography!)

Suppose Eve discovers Alice's request and intercepts Bob's reply, substituting her own public key for Bob's.

Alice sends her message, but it goes to Eve who decrypts the message using her private key!
That could be the end of the story, but more likely. . .

Eve decrypts Alice's "secret" and then uses Bob's public key to. . .
Possibly re-encrypt it and send it to Bob while impersonating Alice. Bob thinks the secret is safe.
Or. . . Substitute a different plaintext and encrypt that and send it to Bob while impersonating Alice.
Or. . . Who knows what Eve might think of.

The moral of the story:
Before a public key can be safely used it must be authenticated as unaltered and belonging to the intended sender.

"Certificate Authorities" meaning trusted third parties, may be used for key authentication. https

For embedded systems, public-key cryptography might just be too much of a computing burden.
One use of RSA is to exchange a key for a symmetric cryptosystem. Then the "real" plaintext is sent via a more computationally efficient cryptosystem.

https://commons.wikimedia.org/wiki/File:Public_key_encryption.svg

7

RSA—a type of public-key cryptography

Authentication of a plaintext message using RSA (trade secrecy for authentication)

Bob wants to authenticate a message he will send to Alice.

Bob writes the plaintext and then uses is private key to encrypt it. He sends both the plaintext and the cyphertext to Alice. Alice can use Bob's public key to decipher the cyphertext portion of the message. If it matches the plaintext portion, then Bob is the only one who could have sent it because only Bob has the private key to match his public key. Alice is also be assured that the plaintext has not been corrupted.

One problem:
If Bob does this more than once and Eve intercepts many messages, she will have a slew of plain-text cypher-text pairs and eventually she will be able, via the use of statistics, to deduce Bob's private key, and his public key is—well—public, so then it is game-over. Eve can impersonate Bob in all respects. There is a fix, which involves the use of a secure hash code—which we have not yet discussed.

(Side-bar item: Authentication via plaintext-cyphertext pairing can also be done with symmetric cryptosystems, with all the same risks, plus also the need to have shared the key in advance.)

https://commons.wikimedia.org/wiki/File:Public_key_encryption.svg

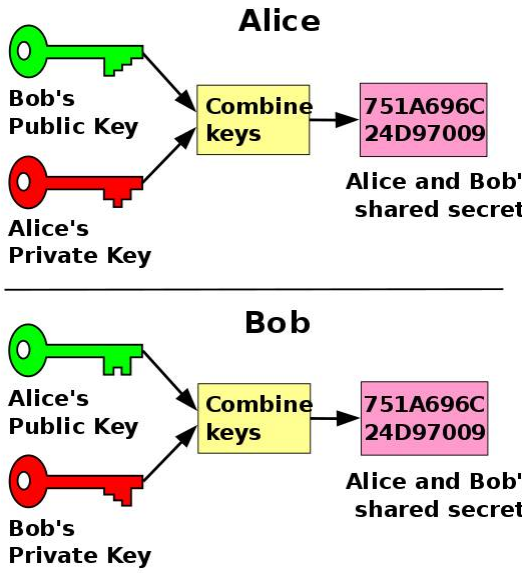
8

RSA—a type of public-key cryptography

Diffie-Hellman Key Exchange
Key for symmetric cryptography exchanged via a public-Key cryptosystem such as RSA

The processing burden of public-key systems is high.
Idea: Use RSA as the secure channel to exchange a key to a symmetric cryptosystem. Then switch over to the symmetric cryptosystem that requires many fewer machine instructions to encrypt or decrypt a message.

All Eve can see are the public keys of Alice and Bob. That's no help for an attack.



https://commons.wikimedia.org/wiki/File:Public_key_encryption.svg

9

Secure Hash Functions

A hash function is a mathematical function that maps many possible inputs to fewer possible outputs.
Parity is an extreme hash function. All inputs get mapped to just logic-1 or logic-0.
In general hash functions can have lots of bits, say 256 bits, so that two messages taken at random have a chance of generating the same hash-function output of only 1 chance in 2^{256}.

We say that

$$h = H(M)$$

Where h = hash function output
M = plaintext
$H()$ is the hash function itself
Notice, no key is involved, and no encryption is achieved.

Now. . . If the hash function $H(M)$ is easy to compute but. . .
plaintext M is virtually impossible to recover from knowledge of h alone, and if
given a hash and message pair $h_1 = H(M_1)$ it is virtually impossible to find a message M_2 such that $h_1 = H(M_2)$
(Pairs such that $H(M_1) = H(M_2)$ may exist—many of them. But they must be virtually impossible to find.)
and if the above properties are true for every possible plaintext M. . .
then you have a **Secure** Hash Function.

10

Secure Hash Functions

Applications:

Passwords

Suppose the OS keeps passwords in a file to verify logons.
This is very risky. Anyone who gets access to the file gets everyone's password.
Some account will have to own the password file.
This owner is typically the system's administrator, a.k.a. the superuser.
The superuser thus can impersonate any account holder.
Maybe you trust the superuser, but should the account get compromised the damage is going to be immense.
The damage will be anonymous! (Unless the attacker is stupid, which happens sometimes.)

Suppose the password file only stores the hash of each password.
To verify a password, hash it and compare the hash to the hash in the password file.
The superuser still owns the password file, but. . .
The hash values will not function as passwords. The superuser *cannot* know the passwords.
I the superuser's account is compromised there still is going to be lots of damage,
but at leas**t** everyone can know that the damage came from the superuser's account.

11

Secure Hash Functions

Applications:

File Verification

A file is transmitted for some purpose that does not require secrecy but does require integrity.
A typical example is the downloading of a program (to assure that malware is not included).
After the file is transmitted its hash code is transmitted via a secure channel, say using RSA.
The received file is re-hashed at the destination. That result is compared to the securely received hash.
If there is a mismatch the downloaded file is discarded.

Authentication of public-key encrypted messages.

RSA is subject to a man-in-the-middle attack on public key exchange. (Described on a previous page.)
Scenario: Alice wants to send a secret to Bob using RSA (or any public-key cryptosystem).
Alice requests in plaintext Bob's public key from Bob
Bob sends his public key in plaintext. He also hashes it. He encrypts the hash with his *private* key.
Bob also sends the cyphertext of the hash, perhaps by appending it to the public key.
Eve intercepts. But she is foiled. She can substitute her public key and hash it,
but she cannot encrypt her hash with Bob's private key.
Suppose she passes the message from Bob unaltered to Alice to remain undiscovered.
Alice receives Bob's public key. She uses that to decrypt the cyphertext hash that was also sent.
The she re-hashes Bob's public key as received in plaintext.
If the re-hashed hash matches the decrypted hash, then she knows she really
does have Bob's public key.
In all this Eve has gained nothing of value. Bob's public key, sent in plaintext, was public to begin with.

12

Secure Hash Functions

Applications:

Spam, denial-of-service, brute-force abatement

Some messages require trivial expense to send.
If the expected value of the message is low on a per-message basis (yet above the cost to send) then the receiver of the message is at risk of abuse.
Examples: spam e-mail, denial-of-service messages, brute-force password attacks
One way to prevent trouble is to raise the cost of the message and thus set a threshold of expected benefit.

Requiring some sort of work before the message is accepted is the normal way to deal with this.
Captchas are common but when automation is desired, they don't work.

Say computer BAD wants to send a message to BANK.
BAD sends message to BANK.
BANK makes up a short random integer h (short = n bits, maybe 10 bits) and sends this back to BAD
BANK requests to be sent a message from BAD that securely hashes to h.
This is a difficult request. On average BAD will have to make up 2^{n-1} random messages M_i
BAD gives up—the amount of work will cost more than it is worth.

Same scenario but computer GOOD is sending to BANK
GOOD will have to make up the required message and pay the price,
it will be a trivial expense relative to the benefit of the message.

13

Broadcast Messages and Timed Release of Keys

The need: Send cyphertext to many recipients at one time.

If a symmetric-key cryptosystem is used many recipients need to be prepared in advance with the key.
This is a risk to security. If there are 100's of copies of the key around, then it is that much easier for the enemy to obtain the key via theft, etc.

If a public-key system is to be used, then he message needs to be encoded over and over again using each recipient's public key and then sent one-at-a-time to the recipients. The main goal is not achieved.

Solution:
This system is more complex than Lee and Seshia make it out to be.
The basic gist of it is that a cypher system is used along with a trusted key server that makes the system time a factor in the key. Messages can be encrypted and broadcast to many users at once. Shortly after the trusted server releases the key via another cryptosystem to only the intended recipients. Keys expire rapidly which makes cracking the system difficult.

14

Cryptographic Protocols

Every cryptosystem has vulnerabilities.
Most vulnerabilities can be avoided by using correct procedures (policies) when using the cryptosystem.
Humans hate policies and often flaunt them, especially in times of high stress.
User training is essential.

Example:
Suppose every text starts with , "Dear <name>," and ends with, "Sincerely Yours, dDB"
In most cryptosystems that is a vulnerability since it allows an enemy to guess the plaintext to match a fraction of the cyphertext. Solution: "Cut the text" in a random place, much as one would cut a deck of cards. The recipient will easily shuffle the text back into the correct order. The enemy will loose track of the match of plaintext with cyphertext.

There are many policies and protocols that must be observed with most cryptosystems to maintain security.
These protocols are much more important than most people give them credit for.

15

Side-channel attacks
Make the system misbehave via some method other than communication.

Examples:
Inject wild variations into the power supply. Some systems will respond badly.
Focus a strong RF field on the system—injects noise into nodes, maybe that makes the machine do something.
Monitor RF emissions—it may be possible to pick up information, information leakage
Monitor power consumption—A classic way for discovery of marijuana growers in some states,
just get a court order to subpoena electric meter billing records.
BTW much more is possible with this than you might expect.
Do unexpected things to the inputs
Dollar bill changer: Attach a pull-string to a bill. Insert bill, get change, pull bill back out of machine.
Repeat. . . (This better not work! But it will probably work if the designer does not think of it.)

Do things with unexpected timing
Vending machine: Insert money, select product. Just as product is beginning to move, pull the change-return lever. Machine stops and change is returned.
Has the product moved enough to jiggle the machine and get the product out?

Privacy and security: Think like a devil. Develop a complete threat model.

16

Message Brokering Services

A message broker is a trusted third party used to facilitate communication.

A message broker can. . .

- Route messages to one or more destinations
- Transform messages to an alternative representation
- Perform message aggregation, decomposing messages into multiple messages and sending them to their destination, then recomposing the responses into one message to return to the user
- Interact with an external repository to augment a message or store it
- Invoke web services to retrieve data
- Respond to events or errors
- Provide content and topic-based message routing using the publish–subscribe pattern

Message brokers are gaining popularity

https://en.wikipedia.org/wiki/Message_broker

17

Message Brokering Services

Becoming popular to include message brokering in the network fabric

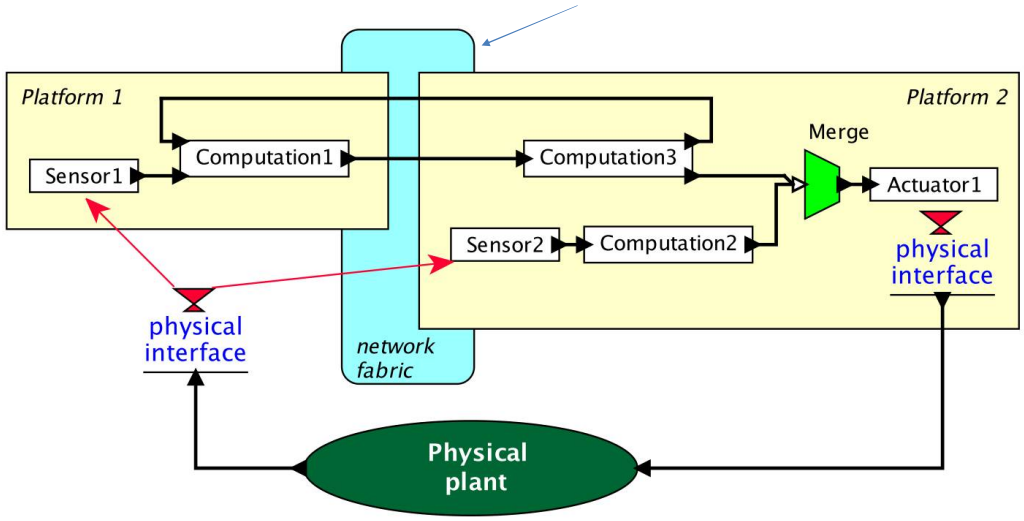


Illustration from our textbook, fair use

18

Message Brokering Services

Standards are emerging.

Example: MQTT (Message Queuing Telemetry Transport)

It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

Use scenario:
River monitoring. Many river-stage sensors report conditions to the broker. The broker organizes the data into a spreadsheet. Users of the data may subscribe to the spreadsheet.

Why not set up your own server for this? Actually, why should you have a server?
Why not have a cloud service do it?
Message brokers future-proof large systems by inserting a point where translation between standards can happen.
Message brokers can anonymize personal data for applications that would be unethical otherwise.

19

Message Brokering Services

The true power of message brokering emerges when many disparate organizations start sharing data.

Smart TVs can report viewer's choices.
Smart phones can report a person’s geographic location.
Nielson or another company that wants to rank videos and TV shows can subscribe to both sets of messages and figure out who is watching the TVs.

Most message brokers offer libraries in various languages (C++, Python, Javascript, etc.)
Most libraries are object-oriented.

20