Principles *of* Security

Douglas Crockford

Yahoo!

White hats vs. black hats.

Security is not hats.

Security is everyone's job.

Don't leave it to specialists.

Things Change

It is not unusual for the purpose or use or scope of software to change over its life.

Rarely are the security properties of software systems reexamined in the context of new or evolving missions.

This leads to insecure systems.

Don’t nobody do nothing stupid and nobody gets hurt.

And this means you.

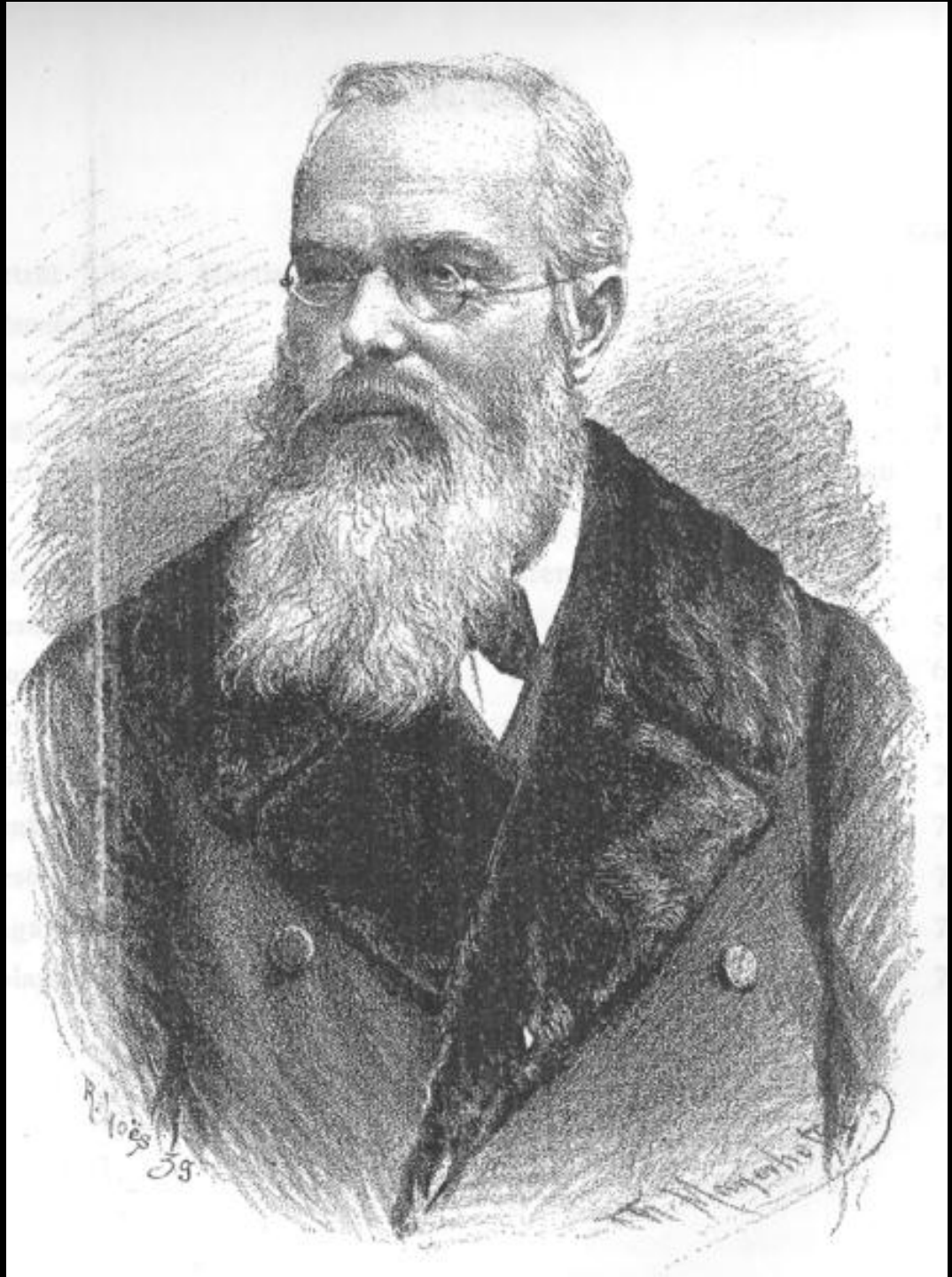
Principles

Not trix and hax.

Deterrence is not effective.

You can't punish an invisible attacker.

Johann Martin Schleyer



Volapük

1880

Debabelization

Jean Guillaume Auguste Victor François Hubert Kerckhoffs

Rebabelization

A secret vice.


JavaScript

Auguste Kerckhoffs

La Cryptographie Militaire

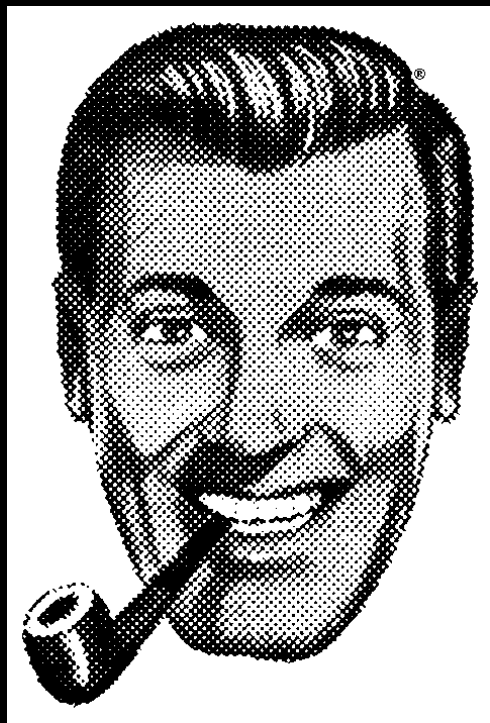
1883

The design of a system should not require secrecy; and compromise of the system should not inconvenience the correspondents.

The Kerckhoffs Principle

Alice

Bob



Key

Key

Plain text

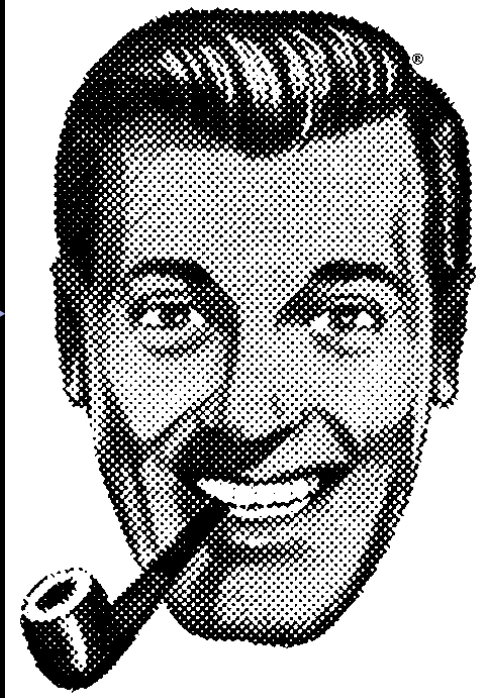
Encrypt

Decrypt

Plain text

Cypher text

Cypher text



There is no security in obscurity.

The more secrets you have,
the harder they are to keep.

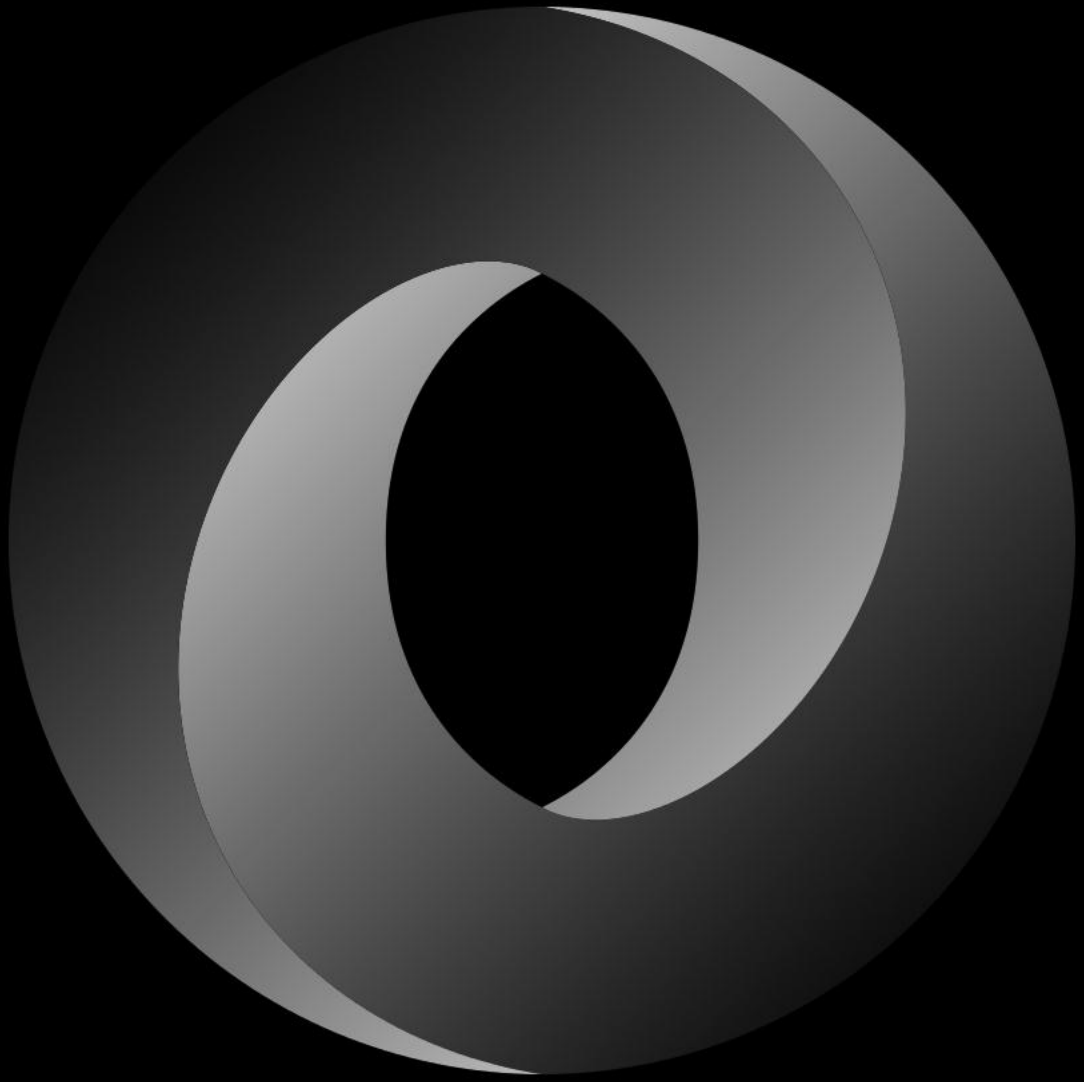
One Time Pad

Truly unbreakable.

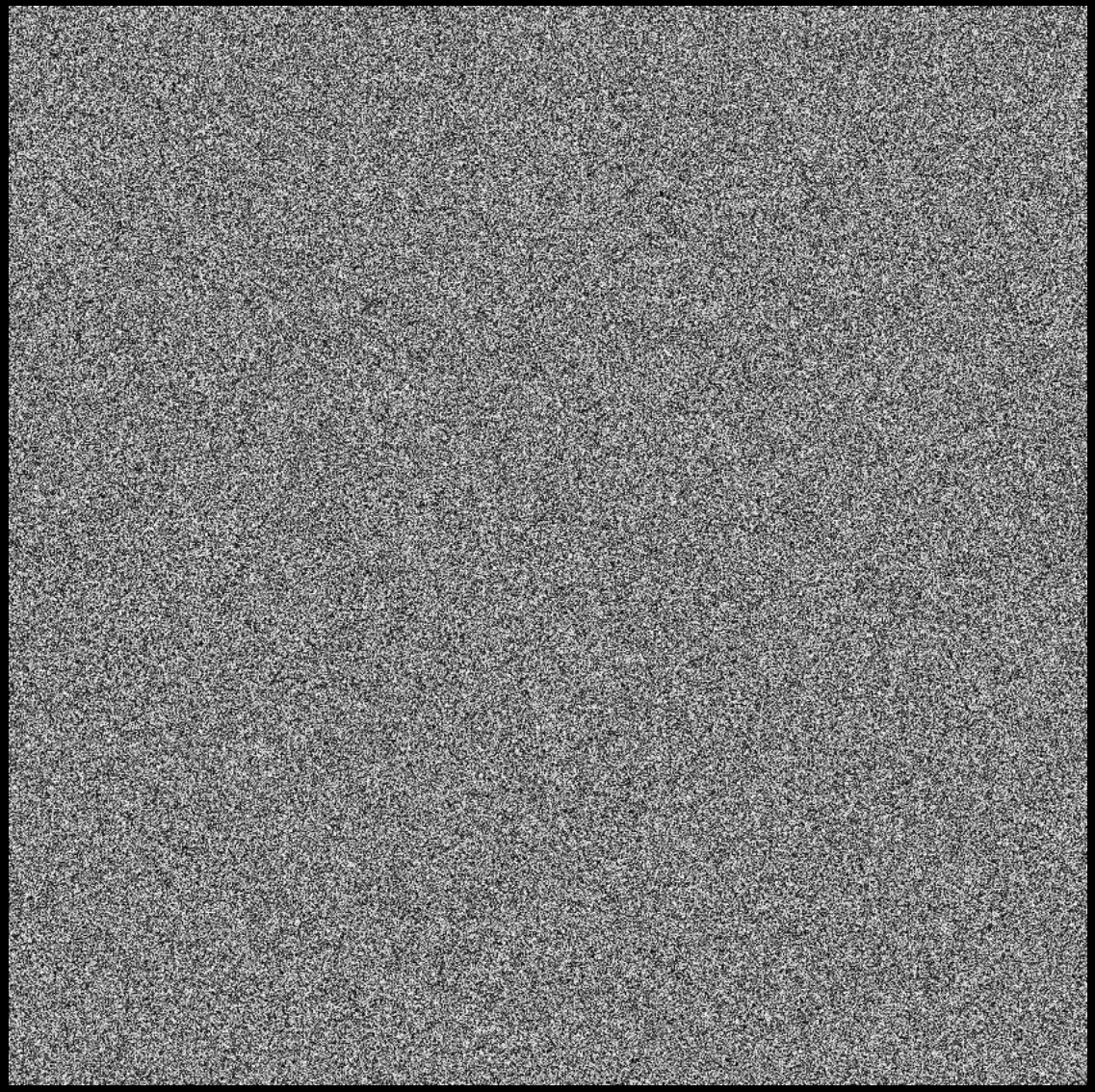
One Time Pad

- The key must always remain secret.
- The key must be at least as long as the plain text.
- The cypher text is obtained by xor of the plain text and the key.

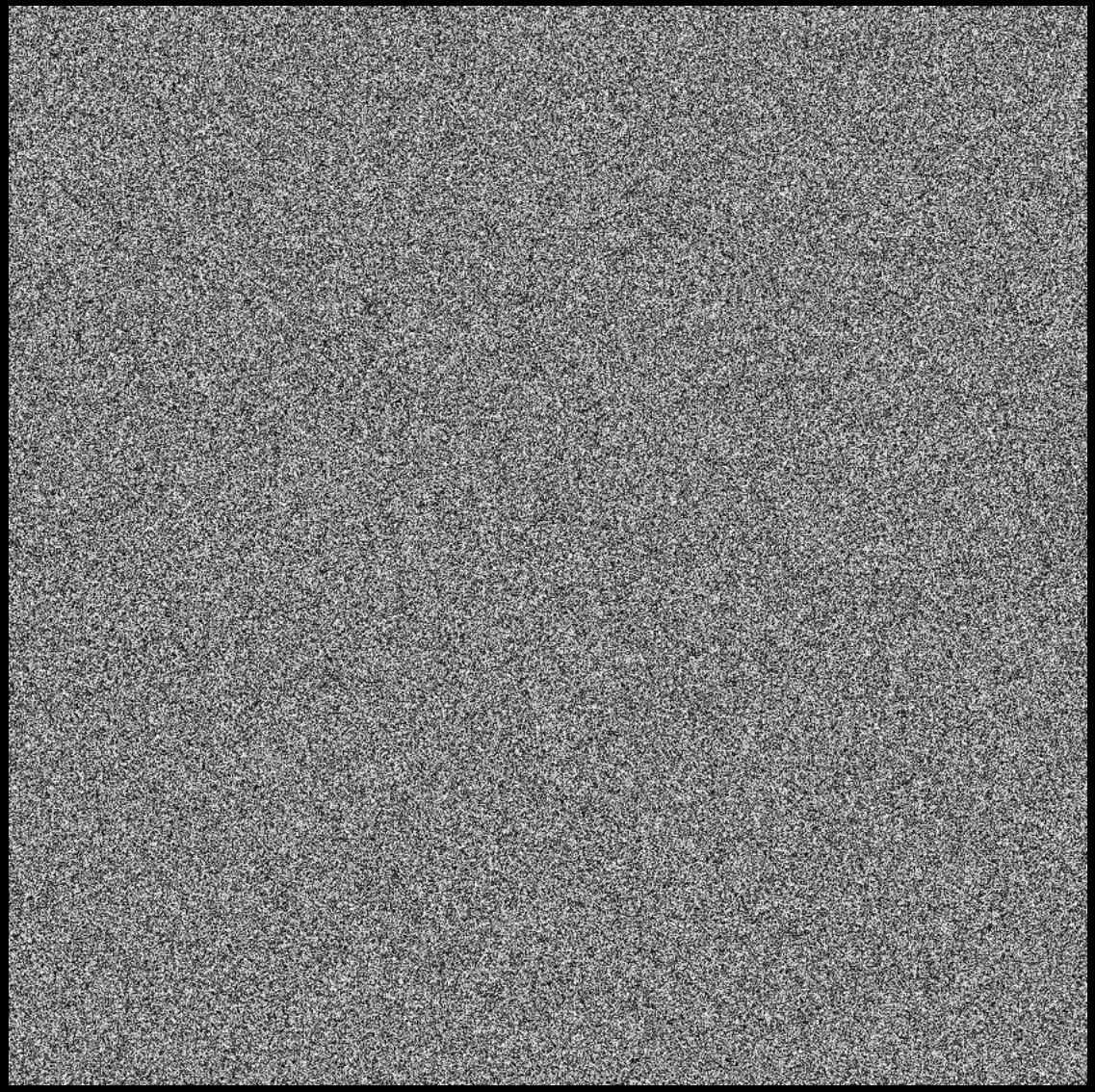
Plain text



Key



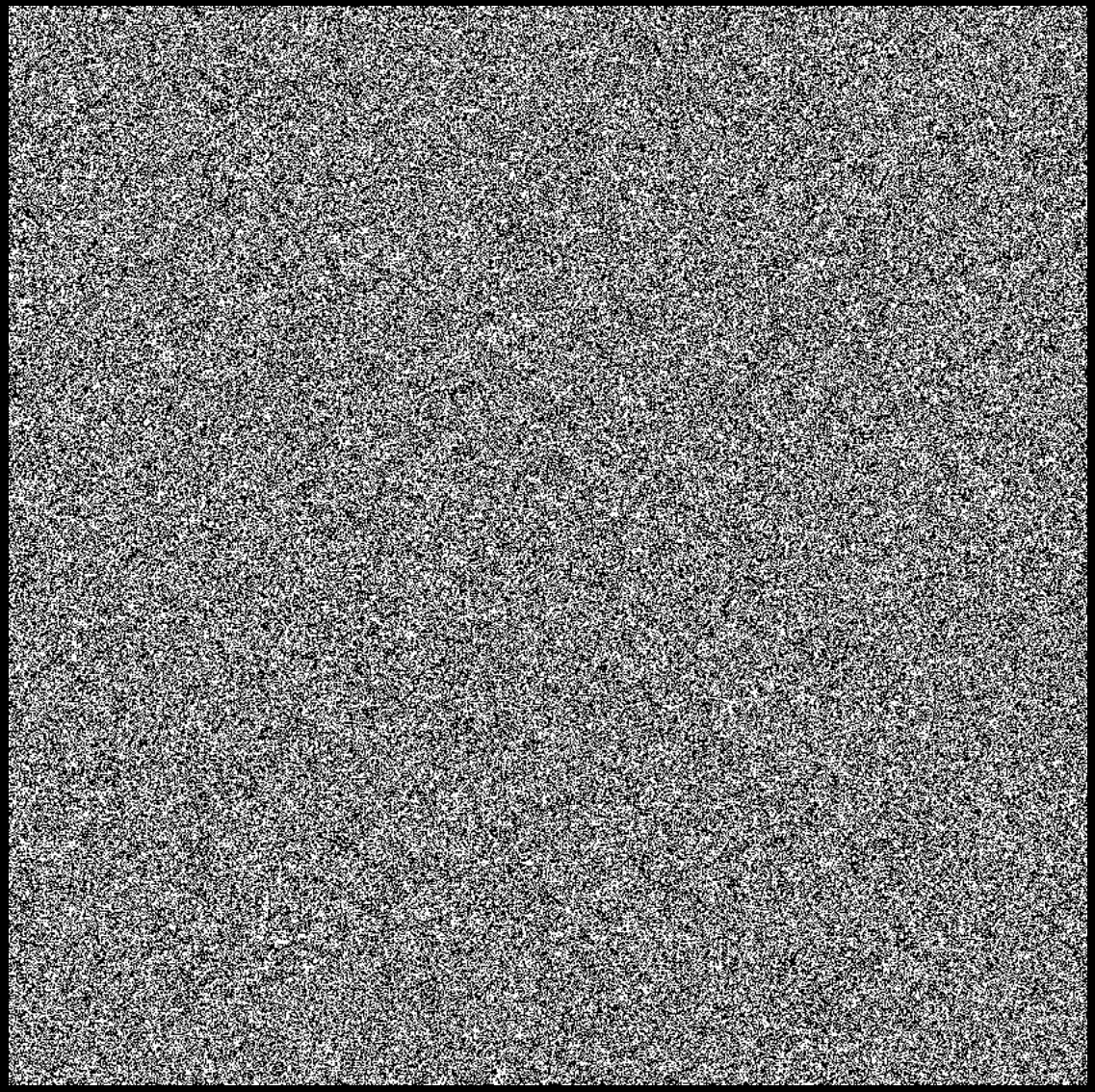
Cypher text



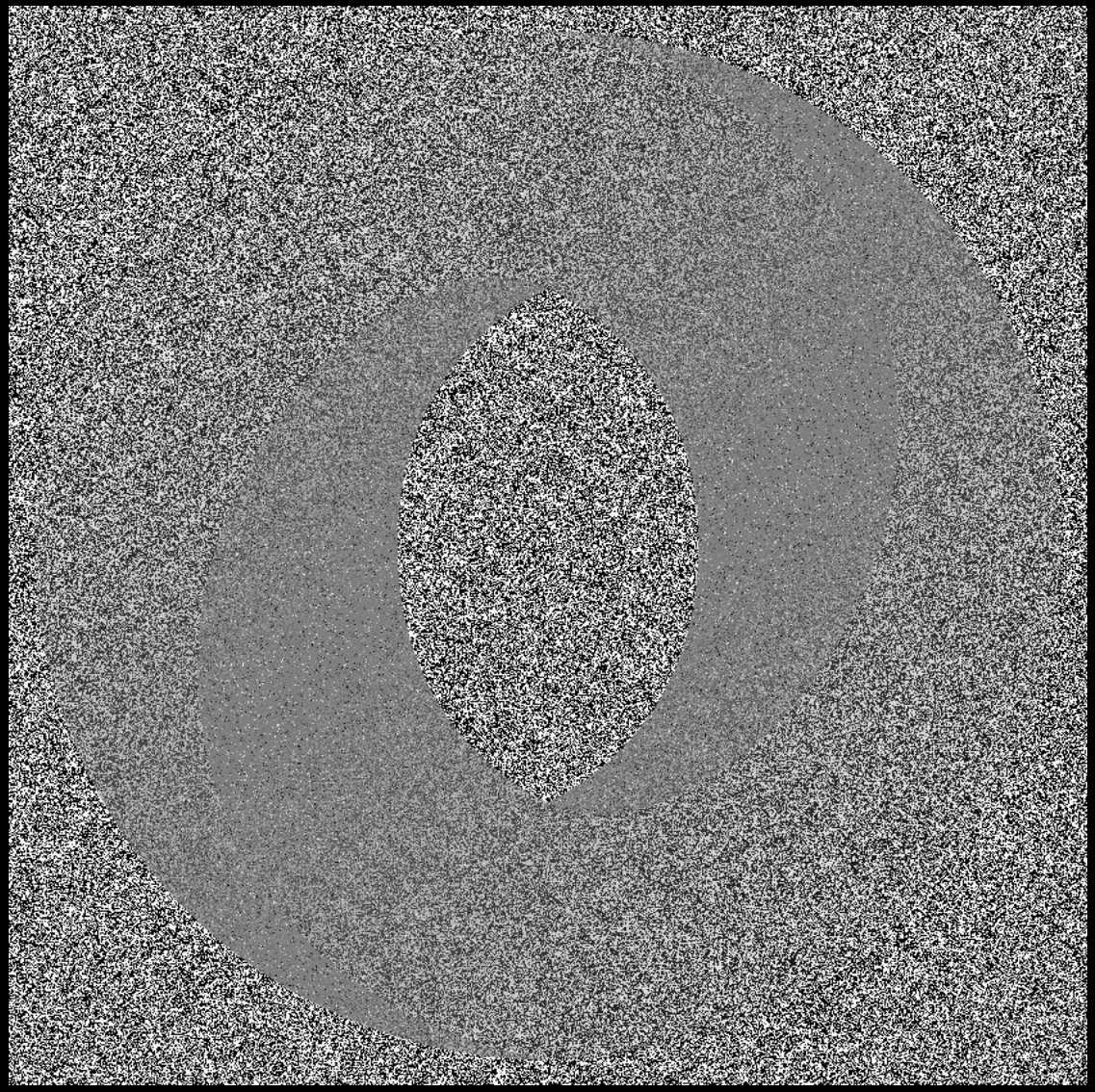
One Time Pad

- The key must always remain secret.
- The key must be at least as long as the plain text.
- The cypher text is obtained by xor of the plain text and the key.
- The key must be perfectly random.

Weak key



Weak cypher text

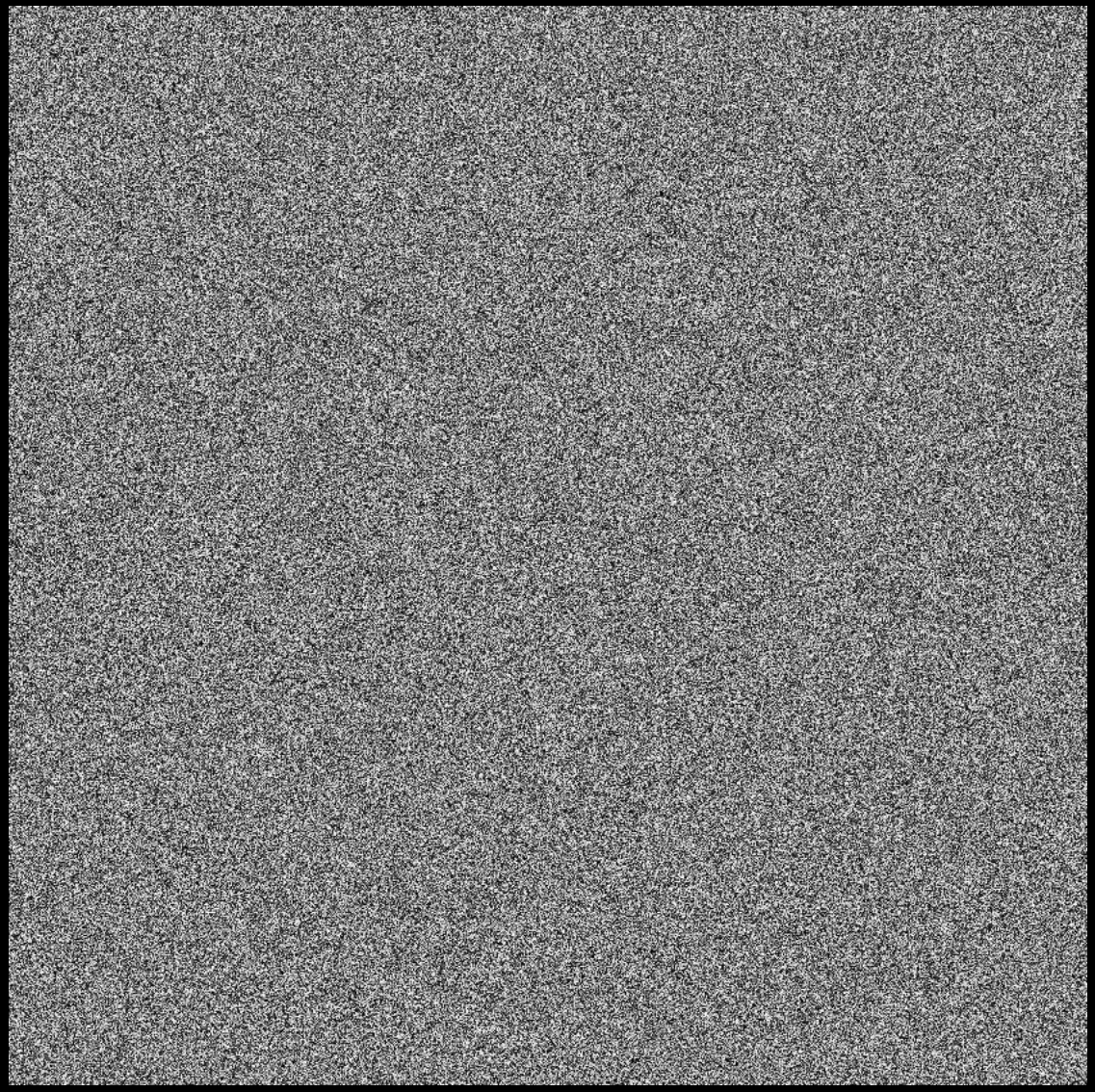


One Time Pad

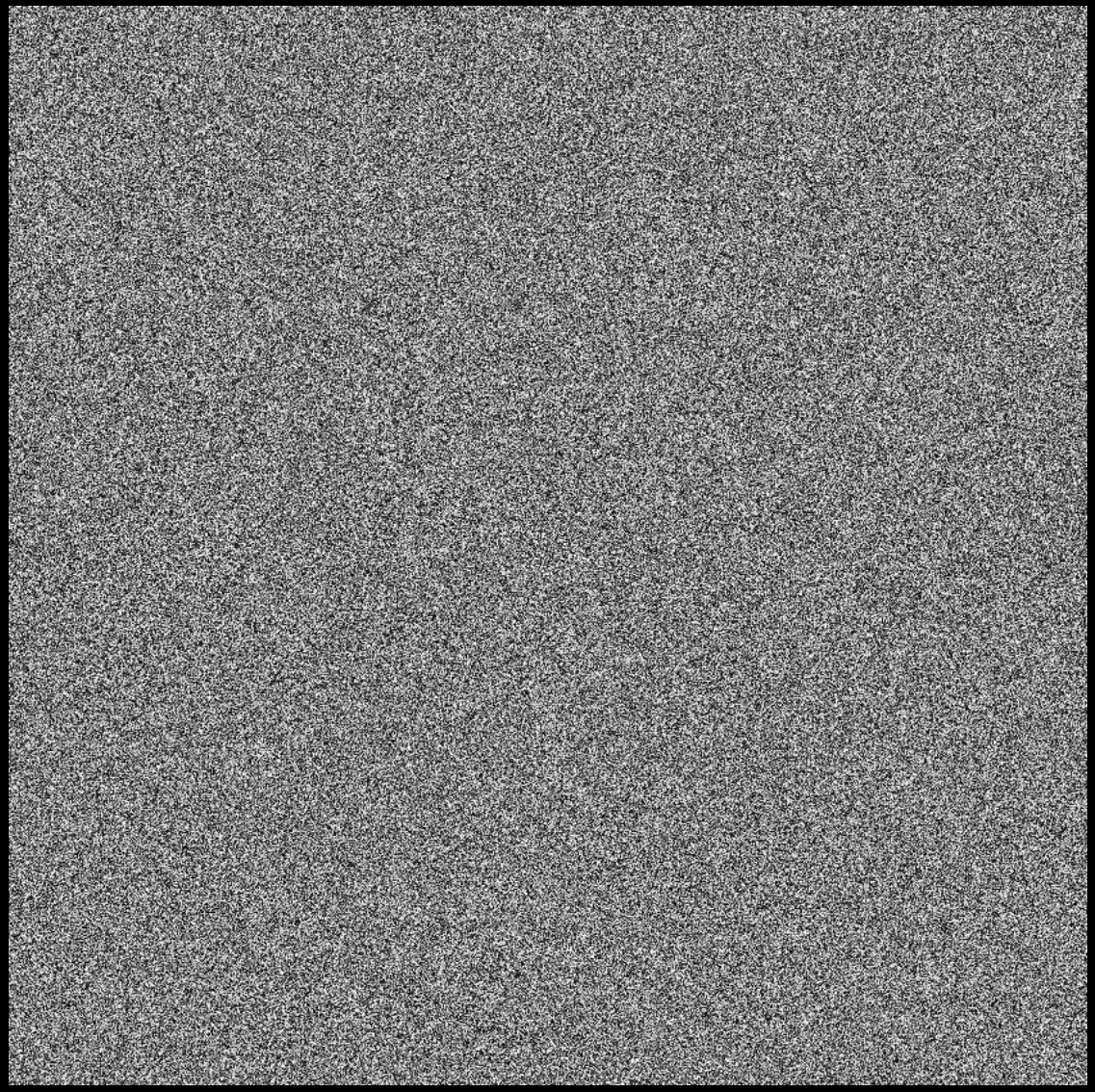
- The key must always remain secret.
- The key must be at least as long as the plain text.
- The cypher text is obtained by xor of the plain text and the key.
- The key must be perfectly random.
- A key must never be used more than once.

Plain text

Reuse key



Cypher text

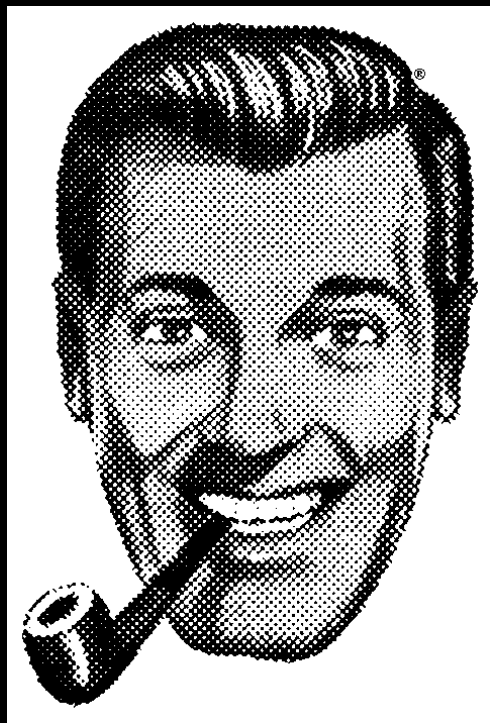


Cypher text xor cypher text

Cryptography is not security.

Alice

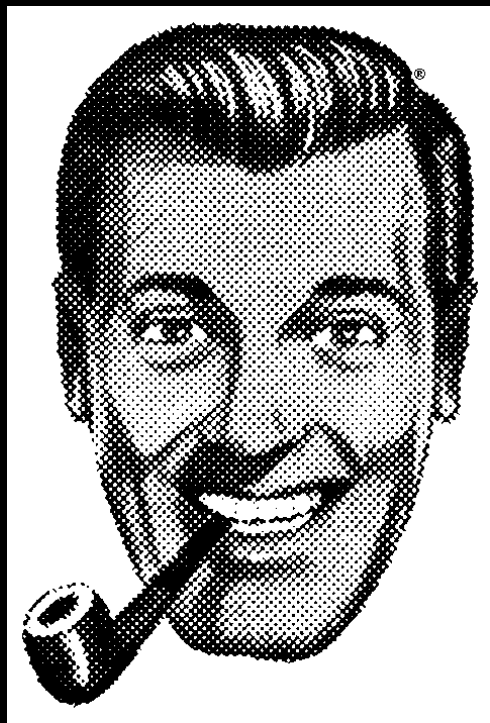
Bob



Alice

Eve

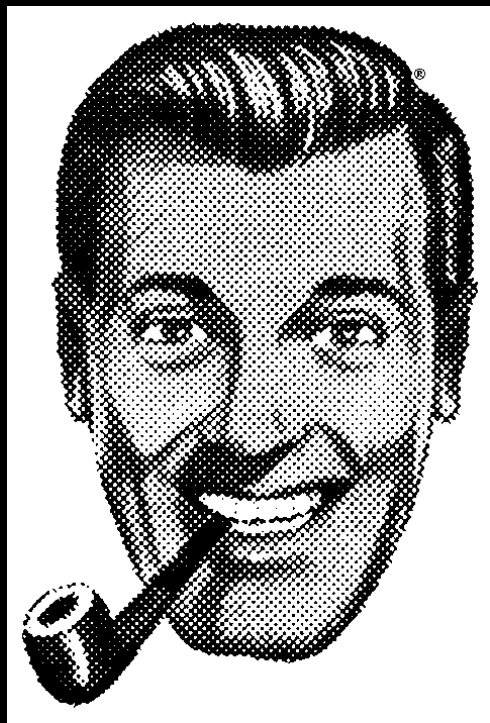
Bob



Alice

Mallory

Bob



Alice

Satan

Security must be factored into every decision.

“We’ll go back and make it secure later.”

**You can't add security,
just as you can't add reliability.**

**Insecurity and unreliability
must be removed.**

Having survived to this point does not guarantee future survival.

The Impossible is not Possible.

If a measure is not effective,
it is ineffective.

Don’t prohibit what
you can’t prevent.

Exploit what you cannot prevent.

False security is worse than no security.

Unnecessary expense
and confusion of risk.

The Browser Platform

- Horribly insecure.
- Still “fixing it later.”
- HTML5 made it worse instead of better.

- It is still better than everything else.

Blame the victim.

Who's interest does the program represent?

The browser got this right.
Every other platform for this wrong.

What the web got wrong

- There can be more interests involved than the user's and the site's.
- A malicious party can exploit coding conventions to inject malicious code.
- That malicious code gets all of the rights of the site.
- This is known as the XSS problem.

What can an attacker do if he gets some script into your page?

An attacker can request additional scripts from any server in the world.

Once it gets a foothold, it can obtain all of the scripts it needs.

An attacker can read the document.

The attacker can see everything the user sees.

An attacker can make requests of your server.

Your server cannot detect that the request did not originate with your application.

If your server accepts SQL queries, then the attacker gets access to your database.

An attacker has control over the display and can request information from the user.

The user cannot detect that the request did not originate with your application.

An attacker can send information to servers anywhere in the world.

The browser does not prevent any of these.

Web standards require these weaknesses.

The consequences of a successful attack are horrible.

Harm to customers.

Loss of trust.

Legal liabilities.

XSS

Cross Site Scripting

Cross site scripting attacks were invented in 1995.

We have made no progress on the fundamental problems since then.

A mashup is a self-inflicted XSS attack.

Advertising is a mashup.

The most reliable, cost effective method to inject evil code is to buy an ad.

Why is there XSS?

- The web stack is too complicated.

  Too many languages, each with its own encoding, quoting, commenting, and escapement conventions.

  Each can be nested inside of each other.

  Browsers do heroic things to make sense of malformed content.

- Template-based web frameworks are optimized for XSS injection.

Why is there XSS?

- The JavaScript global object gives every scrap of script the same set of powerful capabilities.
- As bad as it is at security, the browser is a vast improvement over everything else.

Confusion of Interests

The browser distinguishes between the interests of the user and the interests of the site.

It did not anticipate that multiple interests might be represented.

Within a page, interests are confused.

An ad or a widget or an Ajax library gets the same rights as the site's own scripts.

JavaScript got close to getting it right.

It can be repaired, becoming an object capability language.

HTML

- HTML grants power to confusers.
- HTML is easily confused.
- HTML is forgiving because webmasters were/are incompetent.

- HTML's API, the DOM, is also insecure.

This stuff is not going to get fixed in a hurry.

It is up to the web developer to create secure applications on an insecure platform.

But there is hope…

Any unit of software should be given just the *capabilities* it needs to do its work, and no more.

The Principle of Least Authority

The Actor Model

1973

The Actor Model

- An actor is a computational entity.
- An actor can send messages to other actors only if it knows their addresses.
- An actor can create new actors.
- An actor can receive messages.

- Web workers are actors.
- Web services are not…

Waterken applies the actor model to web services.

Distributed, reliable services.
http://www.waterken.com/

Capability

An address of an actor is a *capability*.

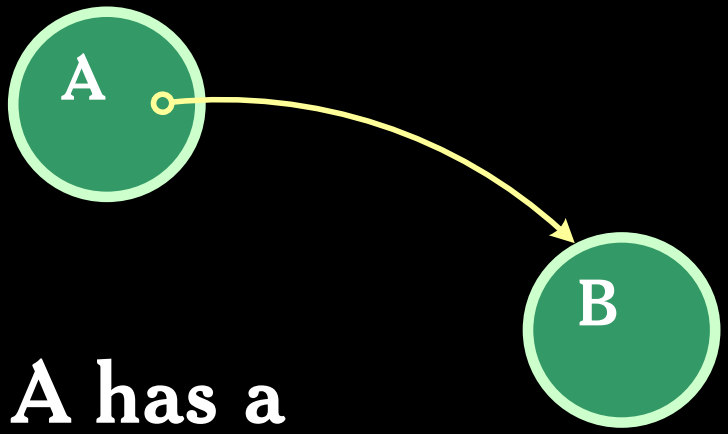
A reference to an object is a *capability*.

An Introduction to Object Capabilities

A is an Object.

Object A has state and behavior.

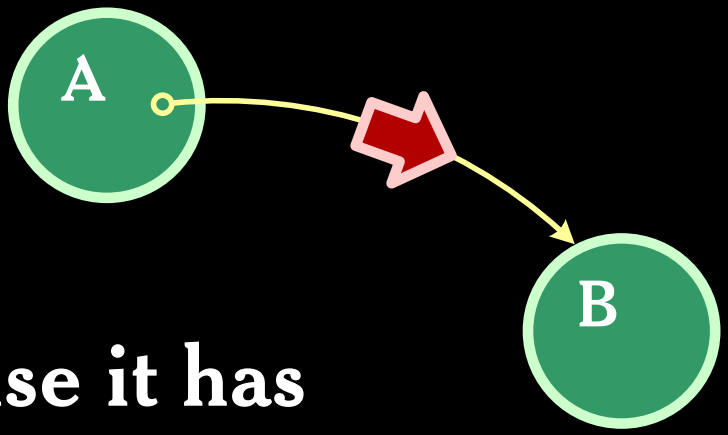
has-a



Object A has a reference to Object B.

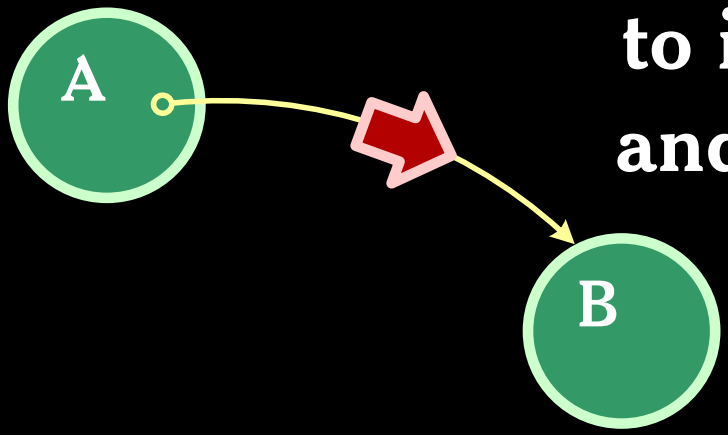
An object can have references to other objects.

Object A can communicate with Object B...



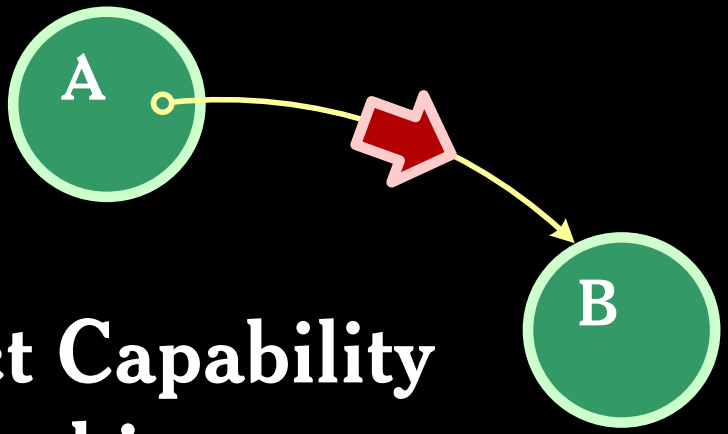
...because it has a reference to Object B.

Object B provides an interface that constrains access to its own state and references.



Object A does not get access to Object B's innards.

Object A does not have a reference to Object C, so Object A cannot communicate with Object C.



In an Object Capability System, an object can only communicate with objects that it has references to.

An Object Capability System is produced by constraining the ways that references are obtained.

A reference cannot be obtained simply by knowing the name of a global variable or a public class.

There are exactly three ways to obtain a reference.

1. By Creation.

2. By Construction.

3. By Introduction.

1. By Creation

If a function creates an object,
it gets a reference to that object.

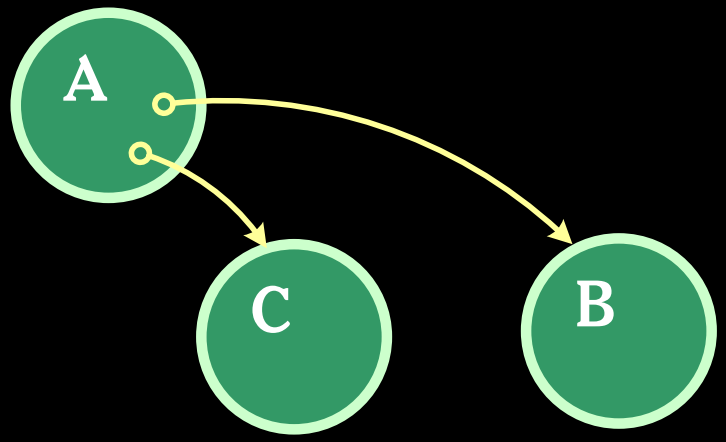
2. By Construction

An object may be endowed by its constructor with references.

This can include references in the constructor's context and inherited references.

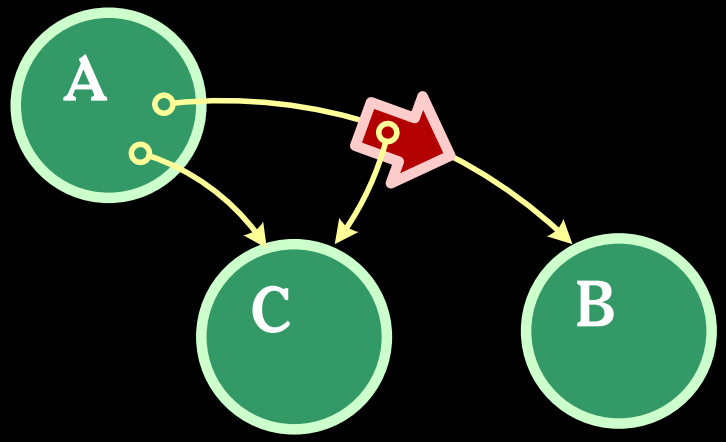
3. By Introduction

A has a references to B and C.
B has no references, so it cannot communicate with A or C.
C has no references, so it cannot communicate with A or B.



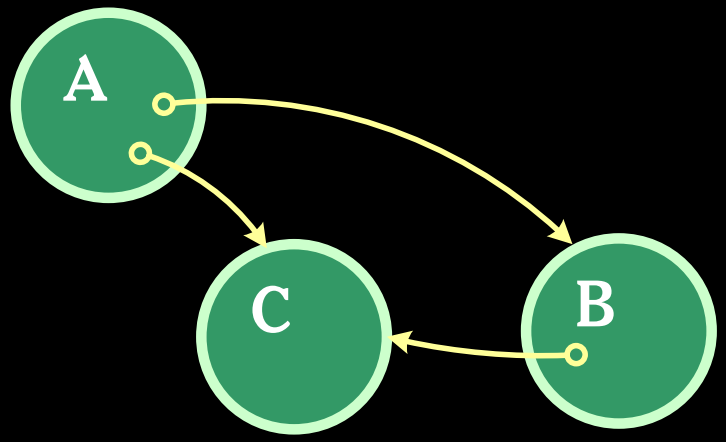
3. By Introduction

A calls B, passing a reference to C.



3. By Introduction

B is now able to communicate with C.



It has acquired the *capability*.

If references can only be obtained by Creation, Construction, or Introduction, then you may have a safe system.

Potential weaknesses include

1. Arrogation.
2. Corruption.
3. Confusion.
4. Collusion.

1. Arrogation

- To take or claim for oneself without right.
- Global variables.
- public static variables.
- Standard libraries that grant powerful capabilities like access to the file system or the network or the operating system to all programs.
- Address generation.
- Known urls.

2. Corruption

It should not be possible to tamper with or circumvent the system or other objects.

3. Confusion

It should be possible to create objects that are not subject to confusion. A confused object can be tricked into misusing its capabilities.

4. Collusion

- It must not be possible for two objects to communicate until they are introduced.
- If two independent objects can collude, they might be able to pool their capabilities to cause harm.

Rights Attenuation

- Some capabilities are too dangerous to give to guest code.
- We can instead give those capabilities to intermediate objects that will constrain the power.
- For example, an intermediate object for a file system might limit access to a particular device or directory, or limit the size of files, or the number of files, or the longevity of files, or the types of files.

Ultimately, every object should be given exactly the capabilities it needs to do its work.

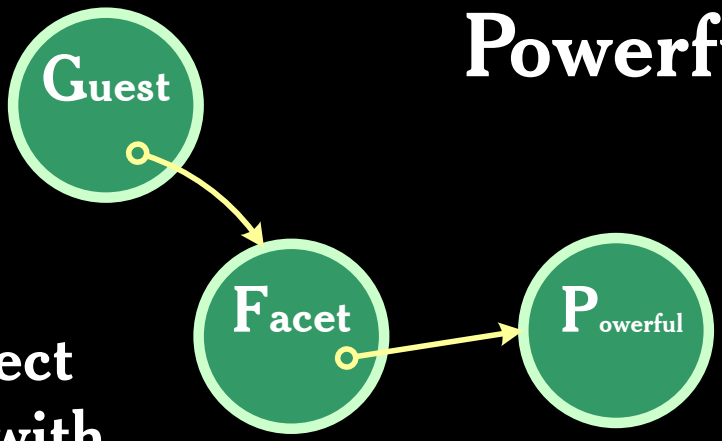
Capabilities should be granted on a need-to-do basis.

Information Hiding - Capability Hiding.

Intermediate objects, or *facets*,
can be very light weight.

Class-free languages can be
especially effective.

The Facet object limits the Guest object's access to the Powerful object.

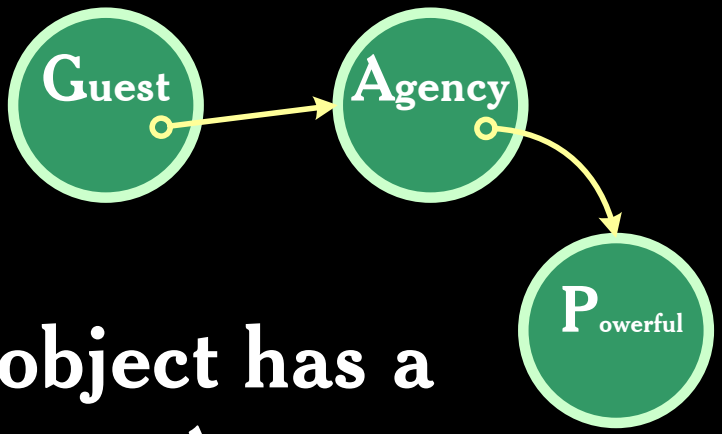


The Guest object cannot tamper with the Facet to get a direct reference to the Dangerous object.

References are not revocable.

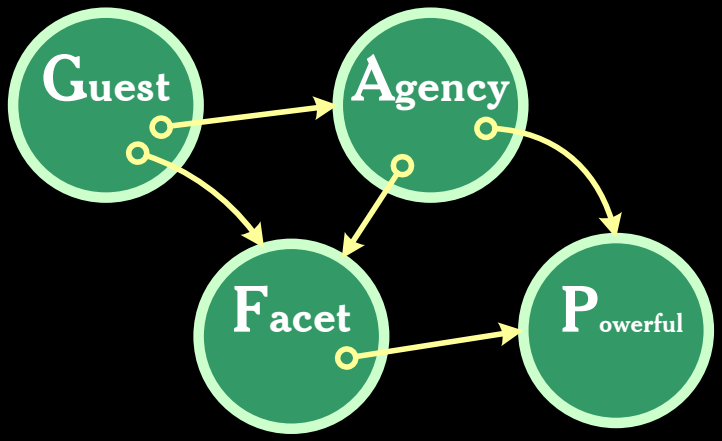
Once you introduce an object, you can't ask it to forget it.

You can ask, but you should not depend on your request being honored.



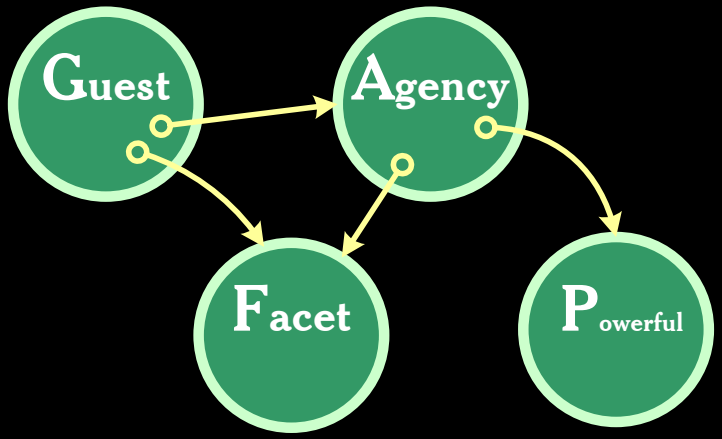
The Guest object has a reference to an Agency object. The Guest asks for an introduction to the Powerful object.

The Agency object makes a Facet, and gives it to the Guest.



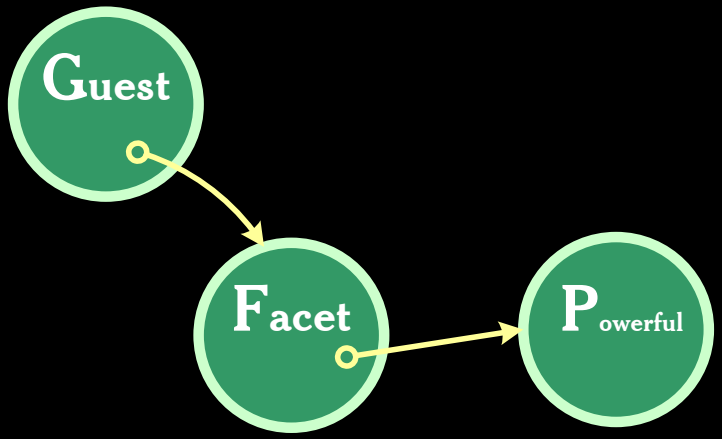
The Facet might be a simple pass through.

When the Agency wants to revoke the capability, it tells the Facet to forget its capability.



The Facet is now useless to the Guest.

A Facet can mark requests so that the Powerful object can know where the request came from.



Facets

- Very expressive.
- Easy to construct.
- Lightweight.
- Attenuation: Power Reduction.
- Revocation.
- Notification.
- Delegation.
- The best OO patterns are also capability patterns

Attenuation is your friend

- Facets can reduce the power of dangerous objects.
- Most code should not be given direct access to innerHTML or document.write.
- Instead of trying to guess if a piece of code can do something bad, give it safe capabilities instead.
- Capabilities can aid in API design.

Function the Ultimate

The Lazy Programmer's Guide to Secure Computing

Marc Stiegler

http://www.youtube.com/watch?v=eL5o4PFuxTY

```
var table = (function () {
   var array = [];
   return {
      get: function (i) {return array[i]; },
      store: function (i, v) {array[i] = v; },
      append: function (v) {array.push(v); }
   };
}());
```

```
var table = (function () {
   var array = [];
   return {
      get: function (i) {return array[i]; },
      store: function (i, v) {array[i] = v; },
      append: function (v) {array.push(v); }
   };
}());
var score;
table.store('push', function () {
  score = this;
});
table.append();
```

Confusion

Confusion aids the enemy.

Bugs are a manifestation of confusion.

With great complexity
comes great confusion.

Keep it simple. Keep it clean.

Code Well

- Good code is ultimately cheaper to produce than bad code, so might as well always write good code.
- Good code is easier to reason about.
- Code that is difficult to reason about is more likely to be problematic.
- Strict conformance to good style rules.
- http://www.JSLint.com/

Never trust a machine that is not under your absolute control.

Don't get more intimate that sharing JSON payloads.

Never trust the browser

- It cannot and will not protect your interests.
- Properly filter and validate all input.
- Properly encode all output.
- Context is everything.
- Filter and encode for the correct context.

Templating and Temporary Insanity

A Simple Attack

```
http://yoursite.com/<script>...</script>


<html><body>
<p>404 File not found:
  <script>...</script>
</p></body></html>
```

- The script runs with the authority of your site.
- The script gets cookies, local storage, everything.

Confusion and Concatenation

Properly encode all of the
non-literal pieces.

A simple attack

- Bad encoding

```
'{"json": "' + xss + '"}'
```

- Bad text

```
xss = '" + alert("XSS") + "'
```

- Good encoding

```
JSON.stringify({json: xss})
```

“Why would anyone do that?”

No leakage

- Do not allow arrogation.
- Everything must be solid. If anything leaks capabilities, all may be compromised.
- If a user has rooted their identity in one of our accounts, and if we leak, the consequences can be tragic.

Inconvenience is not security.

Identity is not security.

Taint ain't security.

Intrusion detection is not security.

Mismanagement

Danog ols e neit gudik.

Danog ols e neit gudik.

Thank you and good night.

- Security is everyone's job.
- Don't nobody do nothing stupid and nobody gets hurt.
- Deterrence is not effective.
- The design of a system should not require secrecy; and compromise of the system should not inconvenience the correspondents.
- There is no security in obscurity.
- Cryptography is not security.
- Security must be factored into every decision.
- You can't add security, just as you can't add reliability.
- The Impossible is not Possible.
- False security is worse than no security.
- Any unit of software should be given just the *capabilities* it needs to do its work, and no more.
- Confusion aids the enemy.
- Never trust a machine that is not under your absolute control.
- Inconvenience is not secuirty.
- Identity is not security.
- Taint ain't security.
- Intrusion detection is not security.
- Security is everyone's job.