Securing Password Storage

Analyzing past failure & secure design

-jOHN
Internal Chief Technology Officer
Principal
@m1splacedsoul

cigital

History: /etc/password

- Circa 1973
- ‘one-way’ password encryption
- chmod a+r etc/password
- DES takes 1 sec. per password

etc/password

```
root:0:0:EC90xWpTKCo
hjackman:100:100:KMEzyulaQQ2
bgoldthwa:101:101:Po2gweIEPZ2
jsteven:102:500:EC90xWpTKCo
msoul:103:500:NTB4S.iQhwk
nminaj:104:500:a2N/98VTt2c
```

History: /etc/password

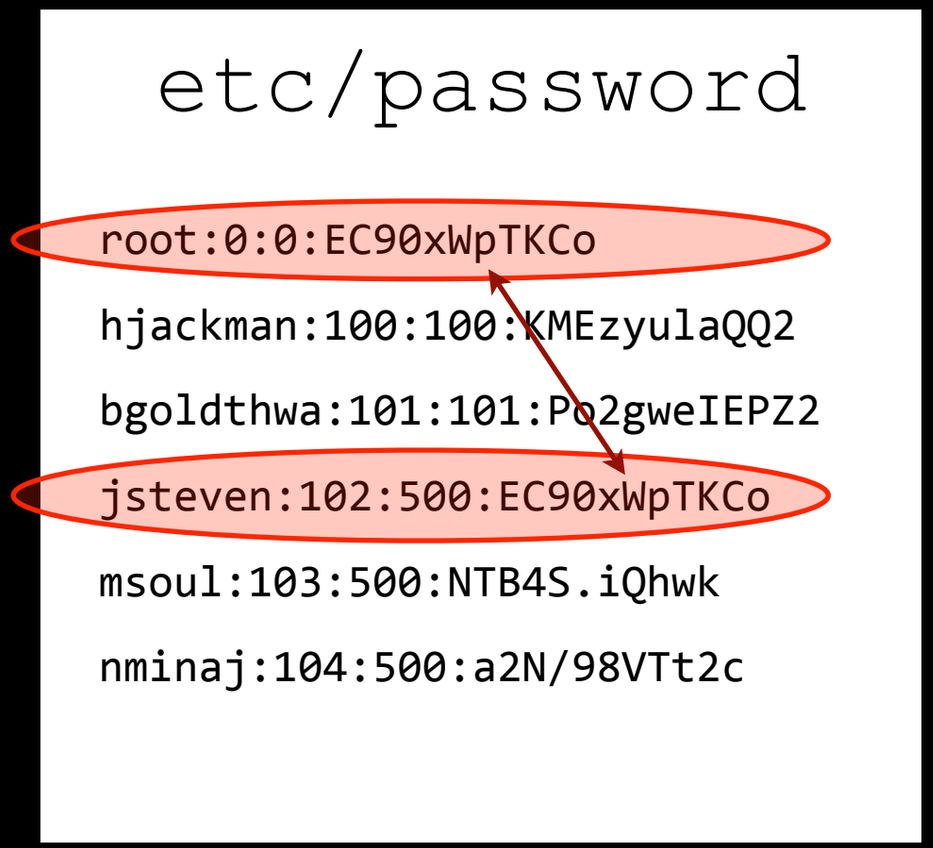
- Circa 1973
- ‘one-way’ password encryption
- chmod a+r etc/password
- DES takes 1 sec. per password

etc/password

```
root:0:0:EC90xWpTKCo
hjackman:100:100:KMEzyulaQQ2
bgoldthwa:101:101:Po2gweIEPZ2
jsteven:102:500:EC90xWpTKCo
msoul:103:500:NTB4S.iQhwk
nminaj:104:500:a2N/98VTt2c
```

History: /etc/password

- Circa 1973
- ‘one-way’ password encryption
- chmod a+r etc/password
- DES takes 1 sec. per password



Salt Control

Fixing /etc/password

I<3NickyMin@j!

Fixing /etc/password

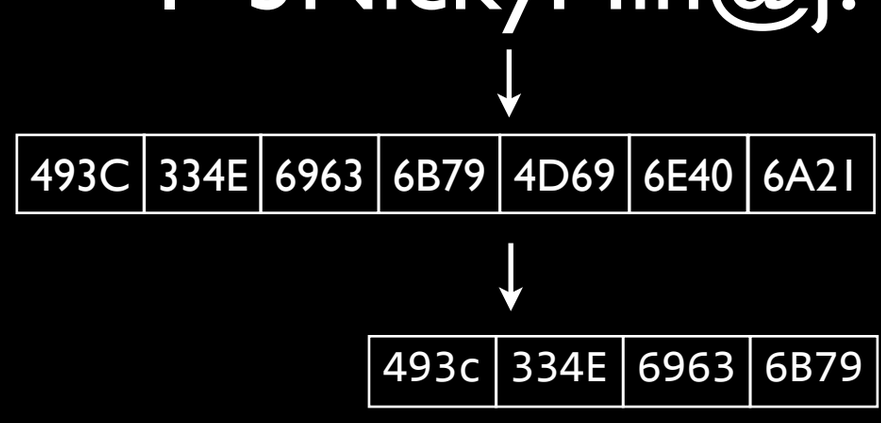
- ‘Good’ Password

I<3NickyMin@j!

112b

Fixing /etc/password

- ‘Good’ Password

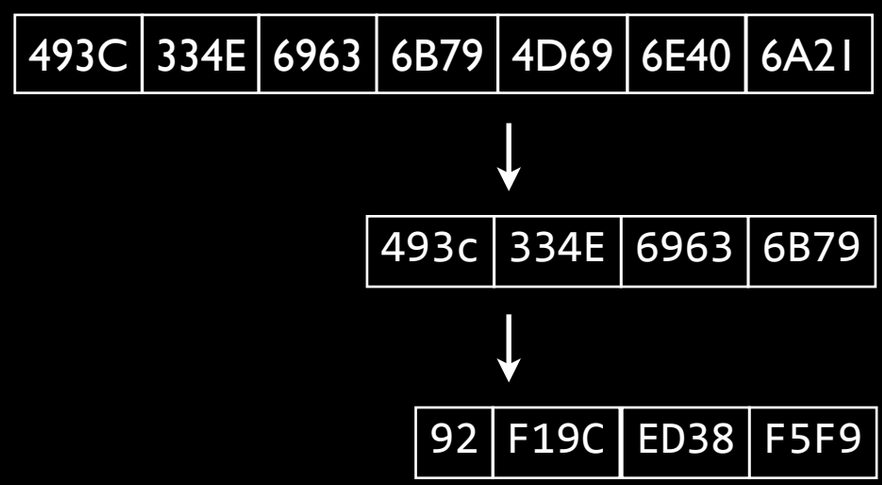


- truncate (8-chars)

Fixing /etc/password

- ‘Good’ Password

I<3NickyMin@j!

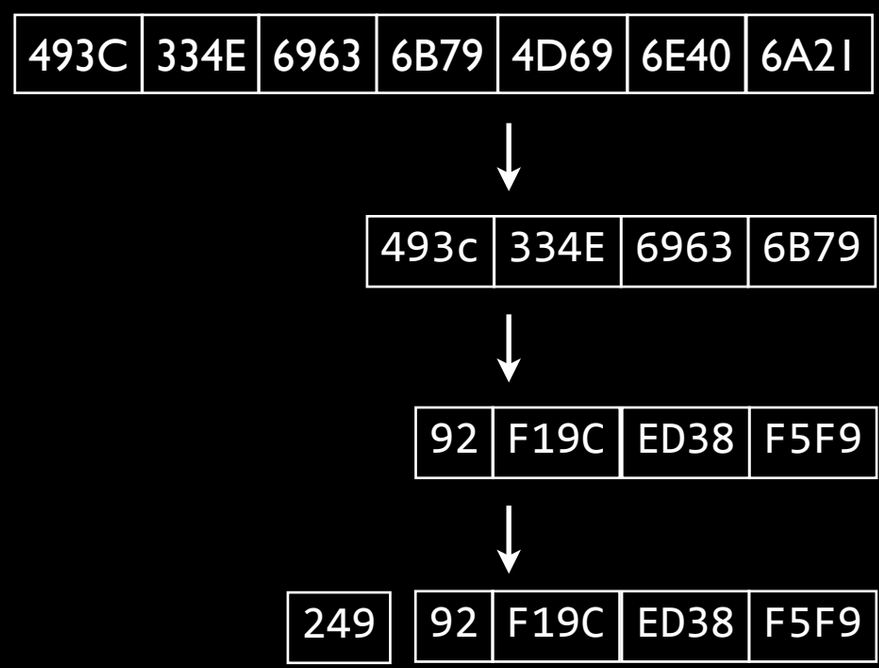


- truncate (8-chars)
- 7bit encoding

Fixing /etc/password

- ‘Good’ Password

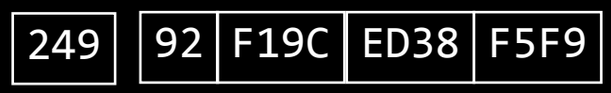
I<3NickyMin@j!



- truncate (8-chars)
- 7bit encoding
- Add 12b SALT

Fixing: /etc/password

- salt + derived key (dK)



68b

Fixing: /etc/password

- salt + derived key (dK)
- plaintext (pT) for encryption

Fixing: /etc/password

- salt + derived key (dK)
- plaintext (pT) for encryption
- crypt(DES, salt, dKey, pT)
  - 25 Iterations
- which is...


```
c_txt = des(s, dK, pT)

for (i=0; i<25; i++)

     c_txt = des(s, dK, c_txt)
```

Brute-forcing Nicki

I<3NickyMin@j!

Brute-forcing Nicki

I<3NickyMin@j! ⟶ ac | EC9 | 0xWp | TKCo 104b

How much work do I have to do?

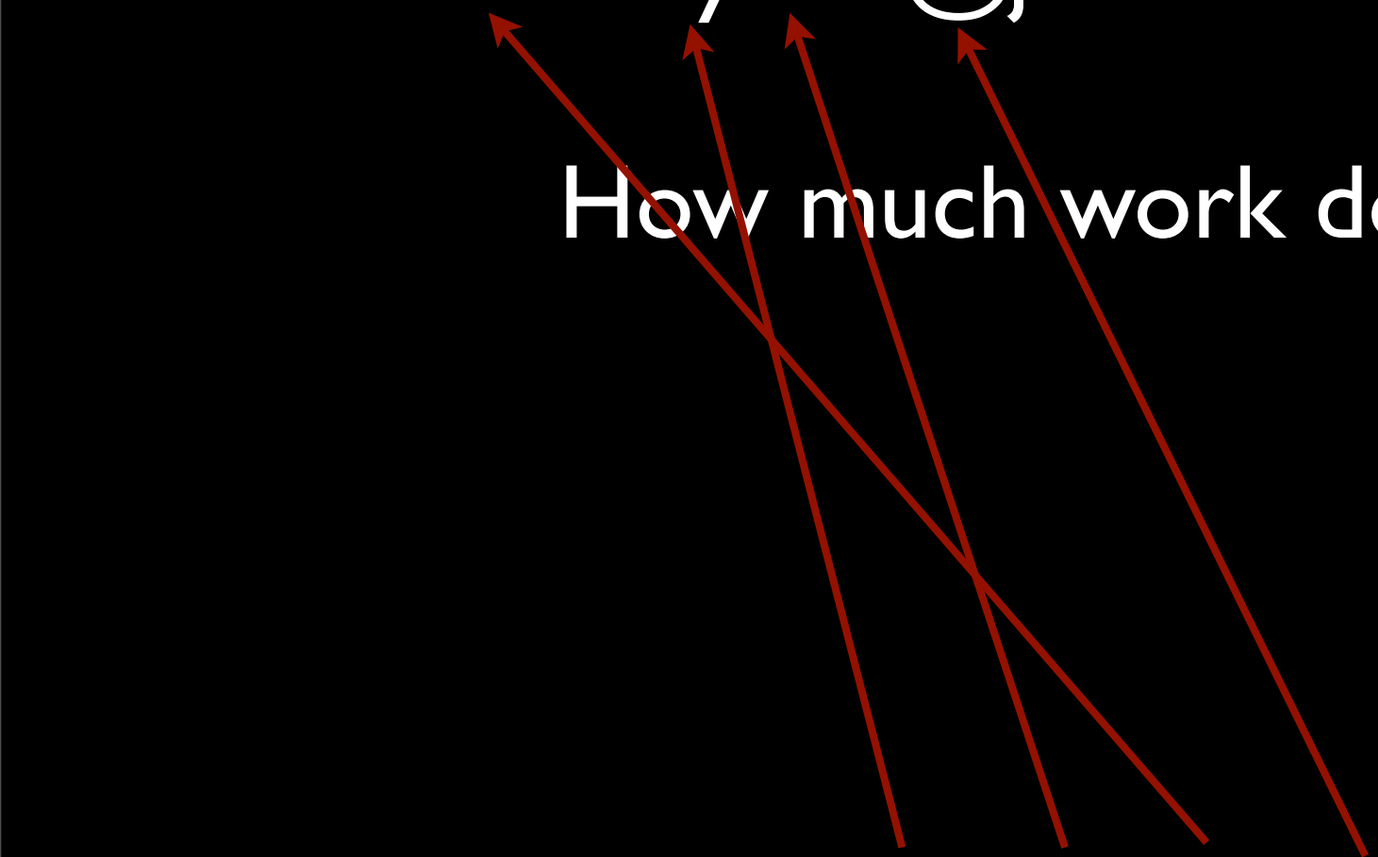
2^{11} 16^{11} 2^{112}

2^{13} 16^{13} 2^{99}

Brute-forcing Nicki

I<3NickyMin@j! ⟶ `ac` `EC9` `0xWp` `TKCo` `104b`

How much work do I have to do?



26+26+10+33 = 95

Brute-forcing Nicki

I<3NickyMin@j! ⟶ `ac` `EC9` `0xWp` `TKCo` `104b`

How much work do I have to do?

26+26+10+33 = 95

95^{15} 2^{99}

Brute-forcing Nicki

I<3NickyMin@j! ⟶ ac EC9 0xWp TKCo 104b

How much work do I have to do?

2^{112}

26+26+10+33 = 95

95^{15} 2^{99}

...Bringing us to 2012

What do we have here?

00000fac2ec84586f9f5221a05c0e9acc3d2e670
0000022c7caab3ac515777b611af73afc3d2ee50
deb46f052152cfed79e3b96f51e52b82c3d2ee8e
00000dc7cc04ea056cc8162a4cbd65aec3d2f0eb
00000a2c4f4b579fc778e4910518a48ec3d2f111
b3344eaec4585720ca23b338e58449e4c3d2f628
674db9e37ace89b77401fa2bfe456144c3d2f708
37b5b1edf4f84a85d79d04d75fd8f8a1c3d2fbde
00000e56fae33ab04c81e727bf24bedbc3d2fc5a
000005891870 1830b2cca174758f7af4c3d30432
000002e09ee4e5a8fcdae7e3082c9d8ec3d304a5
d178cbe8d2a38a1575d3feed73d3f033c3d304d8
00000273b52ee943ab763d2bb3d83f5dc3d30904

...Bringing us to 2012

What do we have here?

00000fac2ec84586f9f5221a05c0e9acc3d2e670
0000022c7caab3ac515777b611af73afc3d2ee50
deb46f052152cfed79e3b96f51e52b82c3d2ee8e
00000dc7cc04ea056cc8162a4cbd65aec3d2f0eb
00000a2c4f4b579fc778e4910518a48ec3d2f111
b3344eaec4585720ca23b338e58449e4c3d2f628
674db9e37ace89b77401fa2bfe456144c3d2f708
37b5b1edf4f84a85d79d04d75fd8f8a1c3d2fbde
00000e56fae33ab04c81e727bf24bedbc3d2fc5a
000005891870 1830b2cca174758f7af4c3d30432
000002e09ee4e5a8fcdae7e3082c9d8ec3d304a5
d178cbe8d2a38a1575d3feed73d3f033c3d304d8
00000273b52ee943ab763d2bb3d83f5dc3d30904

SHA1('password'): 1e4c9b93f3f0682250b6cf8331b7ee68fd8

What's wrong?

```
SHA1('password'): 1e4c9b93f3f0682250b6cf8331b7ee68fd8
```

00000fac2ec84586f9f5221a05c0e9acc3d2e670
0000022c7caab3ac515777b611af73afc3d2ee50
deb46f05215 2cfed79e3b96f51e52b82c3d2ee8e
00000dc7cc04ea056cc8162a4cbd65aec3d2f0eb
00000a2c4f4b579fc778e4910518a48ec3d2f111
b3344eaec4585720ca23b338e58449e4c3d2f628
674db9e37ace89b77401fa2bfe45614 4c3d2f708
37b5b1edf4f84a85d79d04d75fd8f8a1c3d2fbde
00000e56fae33ab04c81e727bf24bedbc3d2fc5a
0000058918701830b2cca174758f7af4c3d30432
000002e09ee4e5a8fcdae7e3082c9d8ec3d304a5
d178cbe8d2a38a1575d3feed73d3f033c3d304d8
00000273b52ee943ab763d2bb3d83f5dc3d30904

What's wrong?

`SHA1('password'): 1e4c9b93f3f0682250b6cf8331b7ee68fd8`

- If they'd salted, how would I have found *my password*?

00000fac2ec84586f9f5221a05c0e9acc3d2e670
0000022c7caab3ac515777b611af73afc3d2ee50
deb46f05215 2cfed79e3b96f51e52b82c3d2ee8e
00000dc7cc04ea056cc8162a4cbd65aec3d2f0eb
00000a2c4f4b579fc778e4910518a48ec3d2f111
b3344eaec4585720ca23b338e58449e4c3d2f628
674db9e37ace89b77401fa2bfe45614 4c3d2f708
37b5b1edf4f84a85d79d04d75fd8f8a1c3d2fbde
00000e56fae33ab04c81e727bf24bedbc3d2fc5a
0000058918701830b2cca174758f7af4c3d30432
000002e09ee4e5a8fcdae7e3082c9d8ec3d304a5
d178cbe8d2a38a1575d3feed73d3f033c3d304d8
00000273b52ee943ab763d2bb3d83f5dc3d30904

What's wrong?

`SHA1('password'): 1e4c9b93f3f0682250b6cf8331b7ee68fd8`

- If they'd salted, how would I have found *my password*?
- How do I build an attack to reverse *all the passwords?*

00000fac2ec84586f9f5221a05c0e9acc3d2e670
0000022c7caab3ac515777b611af73afc3d2ee50
deb46f052152cfed79e3b96f51e52b82c3d2ee8e
00000dc7cc04ea056cc8162a4cbd65aec3d2f0eb
00000a2c4f4b579fc778e4910518a48ec3d2f111
b3344eaec4585720ca23b338e58449e4c3d2f628
674db9e37ace89b77401fa2bfe456144c3d2f708
37b5b1edf4f84a85d79d04d75fd8f8a1c3d2fbde
00000e56fae33ab04c81e727bf24bedbc3d2fc5a
000005891870 1830b2cca174758f7af4c3d30432
000002e09ee4e5a8fcdae7e3082c9d8ec3d304a5
d178cbe8d2a38a1575d3feed73d3f033c3d304d8
00000273b52ee943ab763d2bb3d83f5dc3d30904

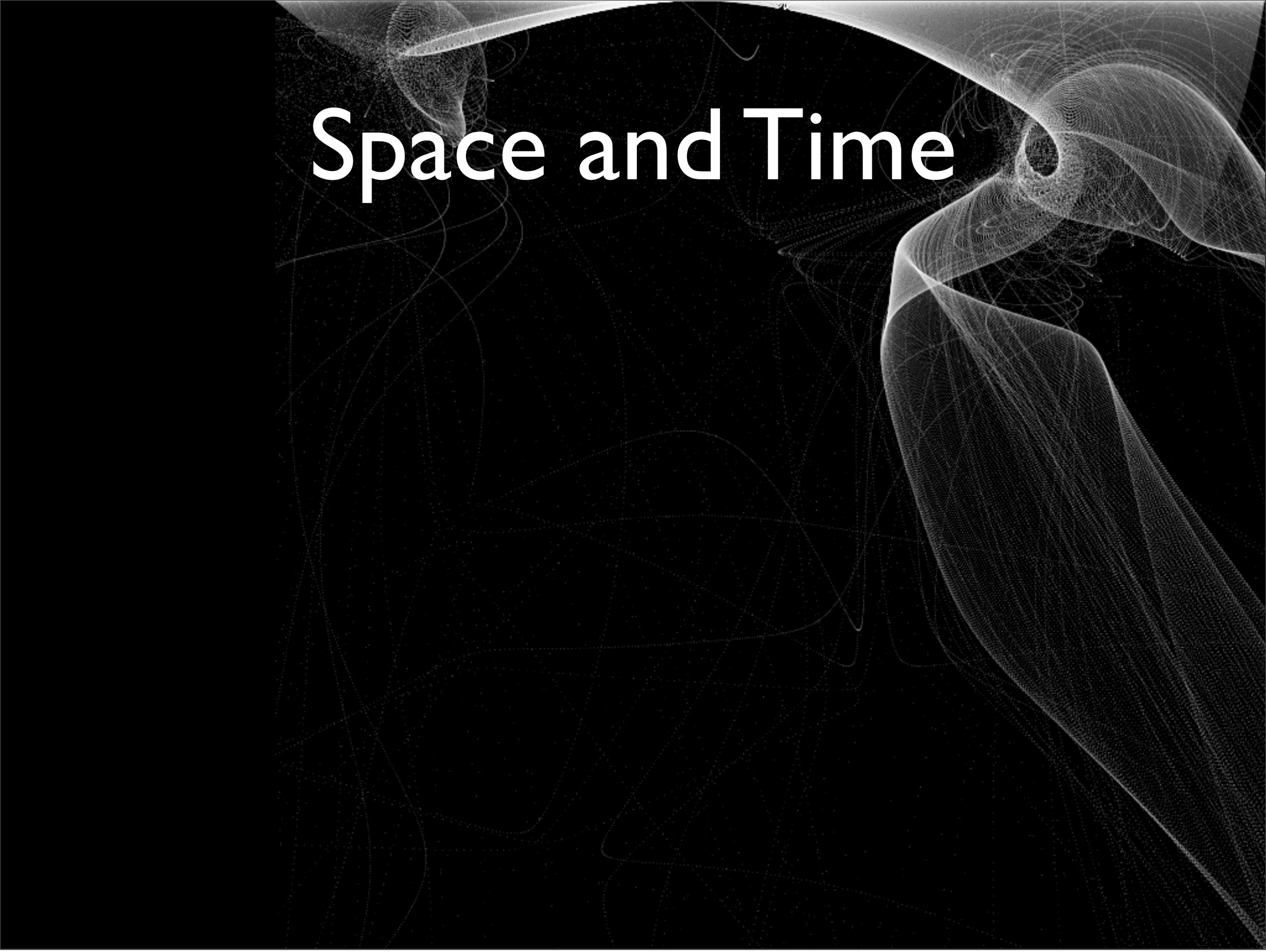
What's wrong?

`SHA1('password'): 1e4c9b93f3f0682250b6cf8331b7ee68fd8`

- If they'd salted, how would I have found *my password*?
- How do I build an attack to reverse *all the passwords?*
- How long *would that take?*

00000fac2ec84586f9f5221a05c0e9acc3d2e670
0000022c7caab3ac515777b611af73afc3d2ee50
deb46f052152cfed79e3b96f51e52b82c3d2ee8e
00000dc7cc04ea056cc8162a4cbd65aec3d2f0eb
00000a2c4f4b579fc778e4910518a48ec3d2f111
b3344eaec4585720ca23b338e58449e4c3d2f628
674db9e37ace89b77401fa2bfe456144c3d2f708
37b5b1edf4f84a85d79d04d75fd8f8a1c3d2fbde
00000e56fae33ab04c81e727bf24bedbc3d2fc5a
000005891870 1830b2cca174758f7af4c3d30432
000002e09ee4e5a8fcdae7e3082c9d8ec3d304a5
d178cbe8d2a38a1575d3feed73d3f033c3d304d8
00000273b52ee943ab763d2bb3d83f5dc3d30904

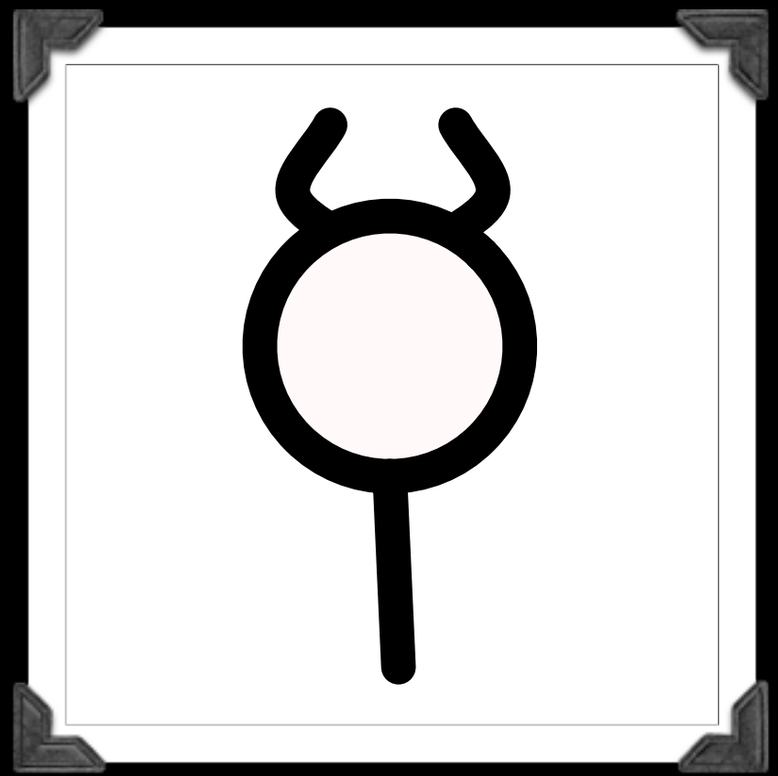
Space and Time



How long does this take?

- Depends on the threat...
  - Some guy
  - Well-equipped Attacker
  - Nation-state

- Is the algorithm supported by your *script-kiddie tool*?



Sir G. Threat IV

circa 2004

Thyme

		(z-z | A-Z | 0-9)^4	(z-z | A-Z | 0-9)^5	(z-z | A-Z | 0-9)^6	(z-z | A-Z | 0-9)^7	(z-z | A-Z | 0-9)^8	(z-z | A-Z | 0-9)^9	(z-z | A-Z | 0-9)^{10}	(z-z | A-Z | 0-9)^{11}
Attacking 2 million hashes (25M/sec)	**NVS 4200M GPU**	1 second	37 seconds	38 minutes	39 hours	101 days	17 years	1,064 years	
	!@#$%^&*()`~-_=+|[]{};:'",.<>/?	4 seconds	5 minutes	8 hours	30 days	7 years	726 years	68,317 years	
Attacking a single hash (32M/sec)	**NVS 4200M GPU**	0.5 seconds	29 seconds	30 minutes	31 hours	80 days	13 years	832 years	
	!@#$%^&*()`~-_=+|[]{};:'",.<>/?	2.5 seconds	4 minutes	6 hours	23 days	6 years	567 years	53,373 years	
Attacking a single salted hash (30M/sec)	**NVS 4200M GPU**	0.5 seconds	31 seconds	32 minutes	33 hours	84 days	14 years	887 years	
6.25% performace loss	!@#$%^&*()`~-_=+|[]{};:'",.<>/?	2.6 seconds	4.1 minutes	6.4 hours	25 days	6.4 years	606 years	56,931 years	
Attacking a single hash (85M/sec)	**$100 Nvidia GTS 250**	0.2 seconds	11 seconds	11 minutes	12 hours	30 days	5 years	313 years	19,413 years
	!@#$%^&*()`~-_=+|[]{};:'",.<>/?	1 second	1.4 minutes	2 hours	9 days	2 years	214 years	20,093 years	1,889,000 years
Attacking a single hash (2.3B/sec)	**$500 ATI Radeon HD 5970**							11.5 years	717 years
	!@#$%^&*()`~-_=+|[]{};:'",.<>/?						7.9 years	743 years	69,803 years

espace

Search Space	Pre-calculated Size
307,000 word dictionary	16 MB
(z-z | A-Z | 0-9)^4	338 MB
(z-z | A-Z | 0-9)6^5	21 GB
(z-z | A-Z | 0-9)^6	1.3 TB
(z-z | A-Z | 0-9)^7	87 TB
(z-z | A-Z | 0-9)^8	5,560 TB
(z-z | A-Z | 0-9)^9	357,000 TB
(z-z | A-Z | 0-9)^{10}	22,900,149 TB

so...

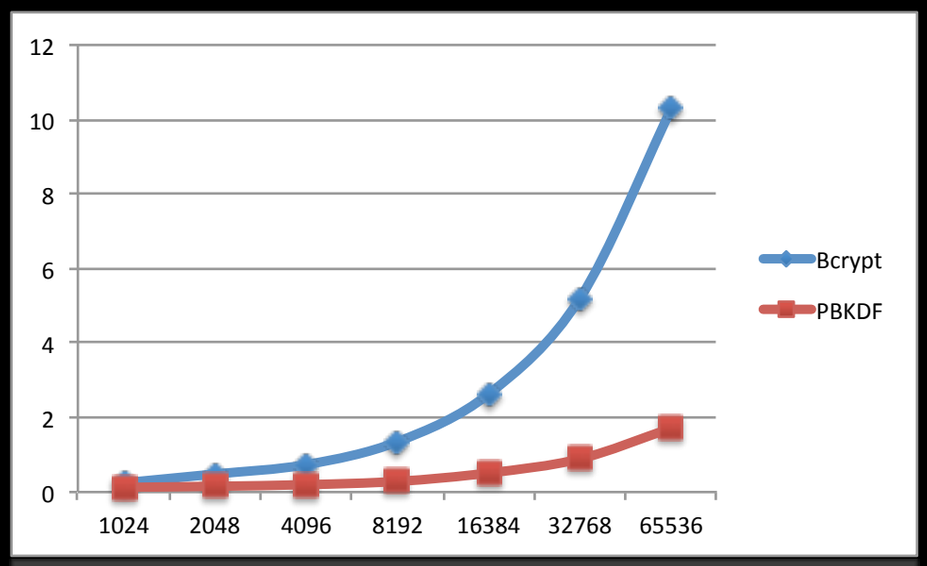
We’re gonna have to brute-force it...

Storing a pre-computed table seems intractable

...OK, how long will this take?

Algorithms scale *linearly* with iterations

Rounds	PBKDF2	Bcrypt
1024	0.125	0.25
2048	0.155	0.483
4096	0.198	0.735
8192	0.286	1.35
16384	0.511	2.632
32768	0.891	5.201
65536	1.715	10.284
(Iteration Count)	(Seconds)	(Seconds)



$$$ vs. Iteration Count

	NVS 4200M	GTX 550TI	GTX 670	GTX 690
Cores	48	192	1,344	3,072
Cost	-	$125	$400	$1,000

Can we fix it?

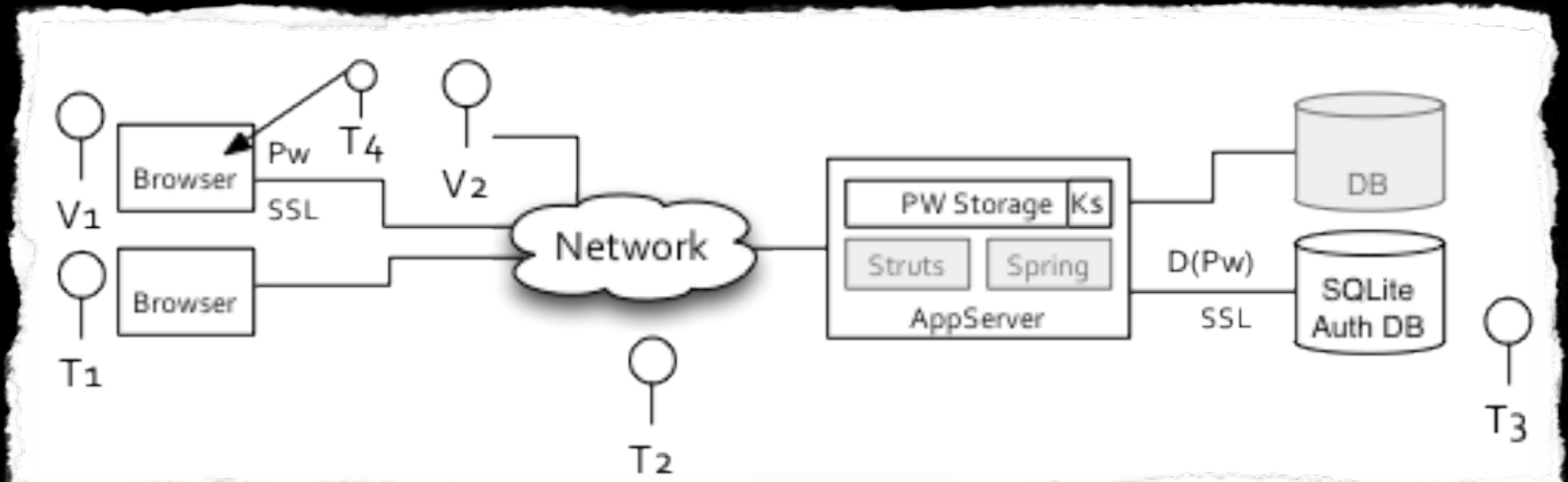
What about those iterations?

c = ?

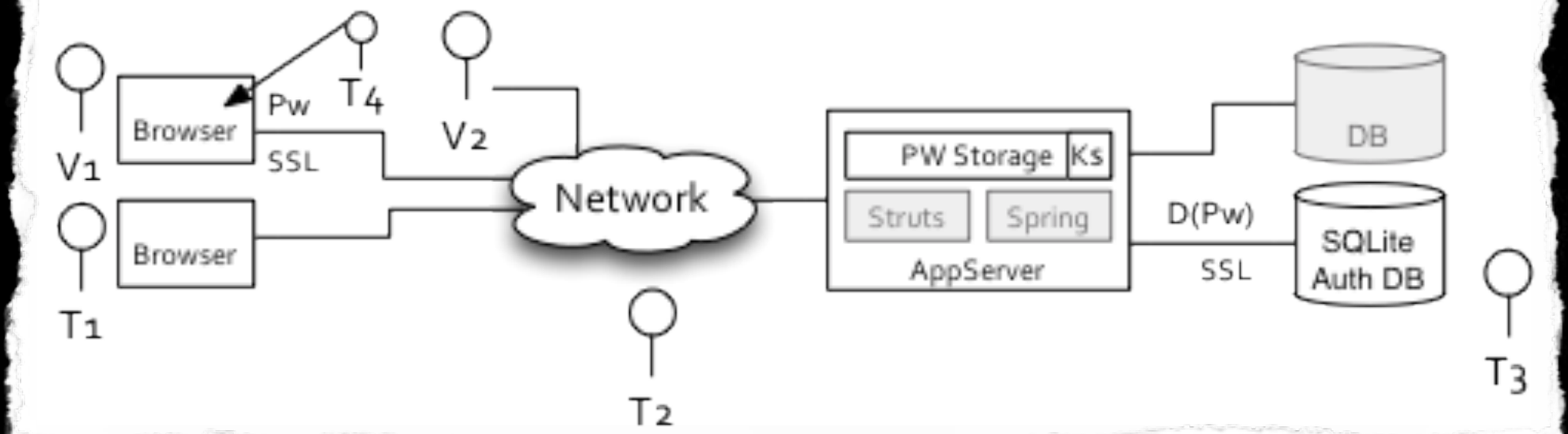
What about those iterations?

OK so look....

Threat Model



Threat Model



1. [V1] - *Active System User* : Compromises one's self through use of the system
    a. Accesses the system normally through a browser
    b. May access the system through a compromised network (exposing them to [T2])

2. [V2] - Offline *System User* : Suffers compromise without use of system
    a. Accesses the system normally through a browser
    b. May suffer compromise due to by duplicate password detection or by bulk export and reversal of stored passwords, for instance.

1. [T1] – *Internet-based Threat* : Access to the app
   - Possesses [1+] valid login to system;
   - Interacts w/ system through browser or AppSec tools;
   - Capable of discovering/executing all AppSec attacks;
   - Capable of acting as MiM [T2] in addition to T1 capabilities;
   - Data show that if [T1] can lift PW, they are likely to also be able to lift/correlate UNs;
   - Capable of rainbow table generation & amateur cryptanalysis;
   - Can NOT conduct state-of-art (accelerated/optimized) cryptanalytic, statistical, or rainbow-table attacks;
   - Can NOT conduct effective bulk phishing, malware installation, or botnet campaigns beyond individual AppSec attacks (i.e. CSRF, etc.).

2. [T2] - *Man-in-the-Middle* : Ability to interpose in communications between victim [V1] and the application server serving it content.
   - May passively observe HTTP traffic;
   - May actively observe and modify HTTP traffic, as a proxy;
   - May passively observe SSL traffic (HTTPS);
   - May be able to interpose, observe, and modify SSL as a proxy;
   - Capable of network-based attacks but not able to 'break' SSL in new or innovative ways;
   - Capable of replaying observed traffic;
   - *** If [T2] conducts [T1.AV3] and modifies code/script bound for [V1], they 'promote' to [T4]. See [T4].

3. [T3] – *LAN-based Threat* : Threat actors within equivalence-class to DB admin
   - May acts as [T2] within the network segment AppServer ← → DB;
   - Has console/network access to AppServer database;
   - May have access (ACLs) to AuthN credentials unless otherwise specified;
   - Presumed to be 'root' on database;
   - May index, sort, and conduct other operations on bulk <protected>(pw) store 'invisibly'.

4. [T4] - *MiB Threat* : [T1] with access to victims' browsers
   - See [T1];
   - Capable of conducting [T1.AV3] (replace code-in-browser);
   - Capable of adding persistent code/data to victims' [V1] browser.

5. [T5] - *Concerted Threat* : [T1]-like threat with capabilities of [T1], [T3], and [T4]. LAN access obtained through means of compromise of other or related systems.
   - Well-funded, patient threat has unlimited time/money
   - Capable of cryptanalytic attack, in addition to more coarse means (such as Rainbow tables)

Threat	Attack Vector	In-Bounds?
[T1] AppSec	AV0 - Observe client operations	Yes
	AV1 - Inject DB, bulk credentials lift	Yes
	AV2 - Brute force PW w/ AuthN API	Yes
	AV3 - AppSec attack (XSS, CSRF, SQLI)	Yes
	AV4 - Register 2 users, compare	Yes
[T2] MiM	AV1 - Interposition, Proxy	No
	AV2 - Interposition, Proxy, SSL	No
	AV3 - Timing attacks	Yes
[T3] Admin	AV1 - Bulk credential export	Yes
	AV2 - [T1] style attack	Yes
	AV3 - Direct action w/ DB	Yes
[T4] MiB	AV1 - Keylogger	No
	AV2 - Other persistent script/data	No
[T5] Concerted	AV1 - PRNG	Yes
	AV2 - DOS	Yes

...Simply it

- Reverse it...
- Chosen plaintext attack
- Dictionary attack
- Brute-force attack
- rainbow table creation
- Length-extension attack
- Oracle Padding attack
- Crypt-analytic attack
- Side-channel attack (such as timing or DPA)

Hash properties

```
digest = hash(plaintext);
```

Hash properties

```
digest = hash(plaintext);
```

- Uniqueness

Hash properties

```
digest = hash(plaintext);
```

- Uniqueness
- Determinism

Hash properties

```
digest = hash(plaintext);
```

- Uniqueness
- Determinism
- Collision resistance

Hash properties

```
digest = hash(plaintext);
```

- Uniqueness
- Determinism
- Collision resistance
- Non-reversibility

Hash properties

```
digest = hash(plaintext);
```

- Uniqueness
- Determinism
- Collision resistance
- Non-reversibility
- Non-predictability

Hash properties

```
digest = hash(plaintext);
```

- Uniqueness
- Determinism
- Collision resistance
- Non-reversibility
- Non-predictability
- Diffusion

Use a better hash?

SHA-2!

- SHA-224
- SHA-256
- SHA-384
- SHA-512

Use a better hash?

SHA-2!

- SHA-224
- SHA-256
- SHA-384
- SHA-512

What property of hashes do these ‘work on’?

hmac properties

```
digest = hash(key, plaintext);
```

hmac properties

```
digest = hash(key, plaintext);
```

- `extends hash` (inherits hash properties)

hmac properties

```
digest = hash(key, plaintext);
```

- `extends hash` (inherits hash properties)
- ‘signature’ : digest/verify by key-holder only

What's the Salt do again?

```
salt || digest = hash(key, salt || plaintext);
```

What's the Salt do again?

```
salt || digest = hash(key, salt || plaintext);
```

- De-duplicates digest texts

What's the Salt do again?

```
salt || digest = hash(key, salt || plaintext);
```

- De-duplicates digest texts
- Adds entropy to input space

What's the Salt do again?

```
salt || digest = hash(key, salt || plaintext);
```

- De-duplicates digest texts
- Adds entropy to input space
- ...raising brute force time

What's the Salt do again?

```
salt || digest = hash(key, salt || plaintext);
```

- De-duplicates digest texts
- Adds entropy to input space
  - ...raising brute force time
  - ...increasing rainbow table size

Adaptive Hashes

```
salt || digest = PBKDF(hmac, salt, pw, c=);
```

Adaptive Hashes

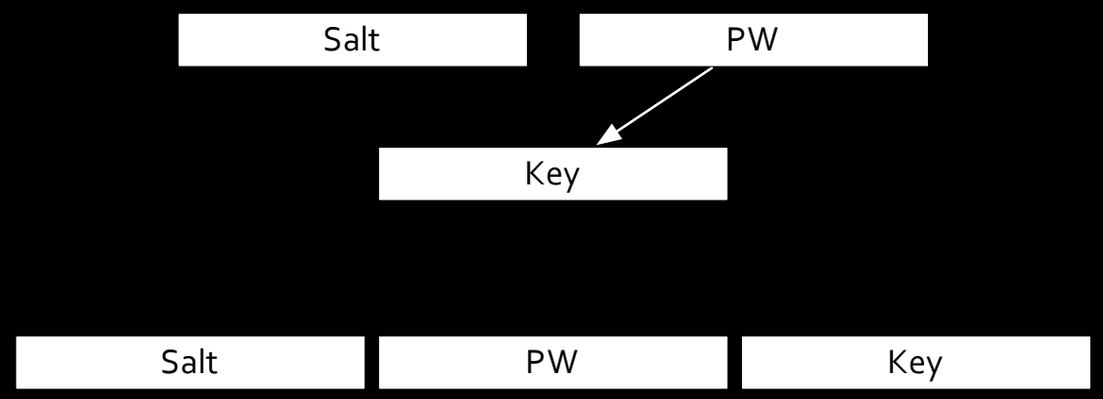
```
salt || digest = PBKDF(hmac, salt, pw, c=);
```

```
prev_round = sha1(key, salt, pw);
for (int i=0; i < c; i++){
    prev_round = sha1(key, salt, prev_round);
}
return prev_round;
```

Adaptive Hashes

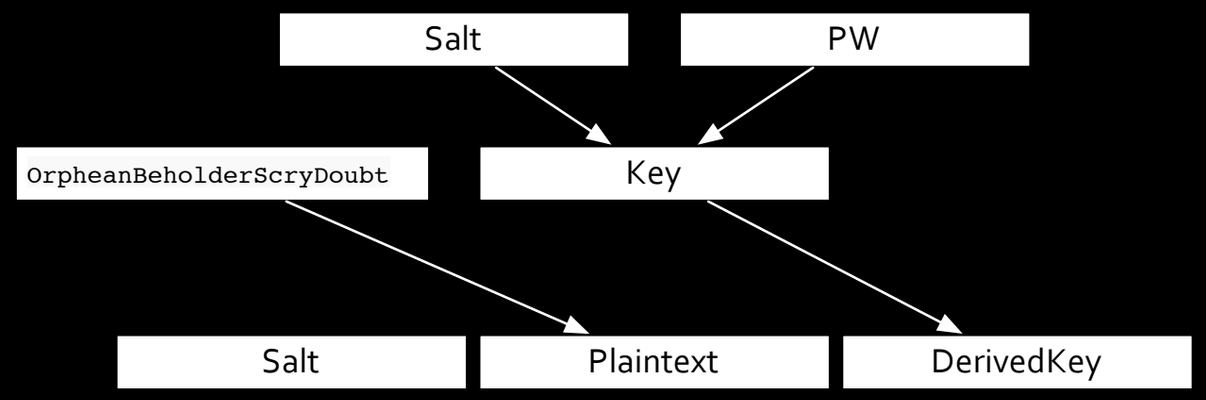
```
salt || digest = PBKDF(hmac, salt, pw, c=);
```

```
prev_round = sha1(key, salt, pw);
for (int i=0; i < c; i++){
    prev_round = sha1(key, salt, prev_round);
}
return prev_round;
```



bcrypt!

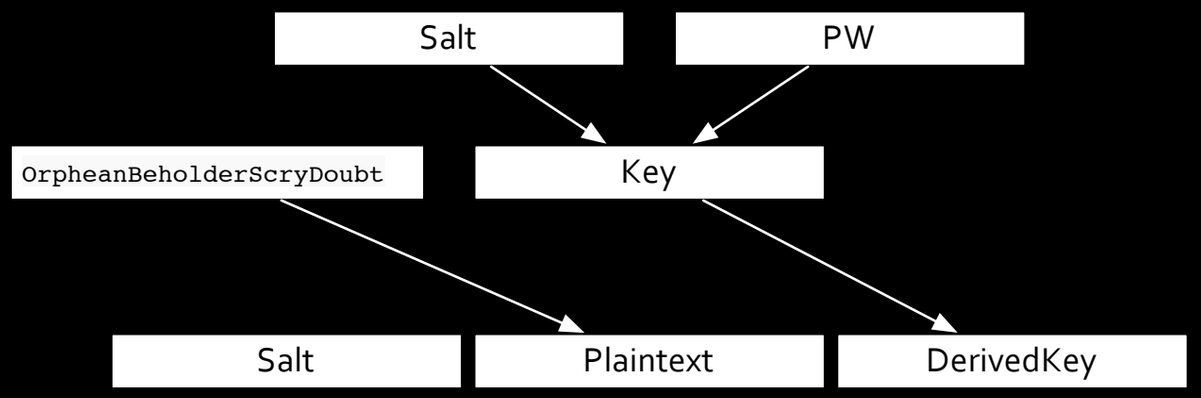
```
salt || digest = bcrypt(salt, pw, c=);
```



bcrypt!

```
salt || digest = bcrypt(salt, pw, c=);
```

```
key = eks(salt, pw);
prev_round = “OrpheanBeholderScryDoubt”;
for (int i=0; i < c; i++){
   prev_round = encryptECB(key, prev_round);
}
return salt||prev_round;
```



Compatible fix

COMPAT, FIPS

<version$_{scheme}$>||<salt$_{user}$>||<digest> := HMAC(<key$_{site}$>, <mixed construct>)

<mixed construct> := <version$_{scheme}$>||<salt$_{user}$>||<pw$_{user}$>

- HMAC := hmac-sha256
- key$_{site}$:= PSMKeyTool(SHA256()):32B;
- salt$_{user}$:= SHA1PRNG() | FIPS186-2():32B;
- pw$_{user}$:= <governed by password fitness>

Fix

Supported (Reversible)

<ciphertext> := ENC(<wrapper key_{site}>, <$version_{scheme}$>||<$salt_{user}$>||<round 1>)

<round 1> := HMAC-2(<key_{site}>, <mixed construct>)

<mixed construct> := <$version_{scheme}$>||<$salt_{user}$>||<pw_{user}>

<key_{site}>:= PSMKeyTool(HMAC-1, <$salt_{site}$>, <pw_{site}>, <c_{inters}>, DkL)

<wrapper key_{site}> := PSMKeyTool(HMAC-1, <$salt_{wrapper}$>, <$pw_{wrapper}$>, <c_{inters}>, DkL)

- ENC := AES
- HMAC-1 := hmac-sha1
- HMAC-2 := hmac-sha512
- PSMKeyTool := rfc2898 PBKDF2(HMAC-1, <$salt_{site}$>, c=10000000, DkL=32B): 32B;
- $salt_{user}$:= SHA1PRNG():32B;
- $salt_{site}$:= SHA1PRNG():32B;
- $salt_{wrapper}$:= SHA1PRNG():32B;
- pw_{user} := <governed by password fitness>

Why? (God why!?)

- Irreversible
- Resists padding / length extension
- Versioned

- No impact to user experience (speed)
- Recovery from stolen key w/o User interaction
- Stolen key AND database still demands brute force
- *Version MAC'd*