New Encryption Primitives for Uncertain Times

Thomas Ristenpart
University of Wisconsin

Covering joint work with:
Scott Coull, Kevin Dyer, Ari Juels, Thomas Shrimpton

Security in our uncertain times:

Iran reportedly blocking encrypted Internet traffic

The Iranian government is reportedly blocking access to websites that use the ...

by Jon Brodkin - Feb 10 2012, 9:44pm IST

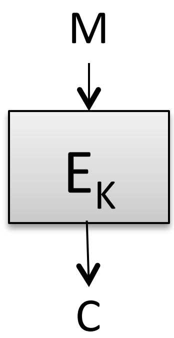
LastPass CEO reveals details on security breach

by Lance Whitney | May 6, 2011 10:19 AM PDT

"Encryption works. Properly implemented strong crypto systems are one of the few things that you can rely on."

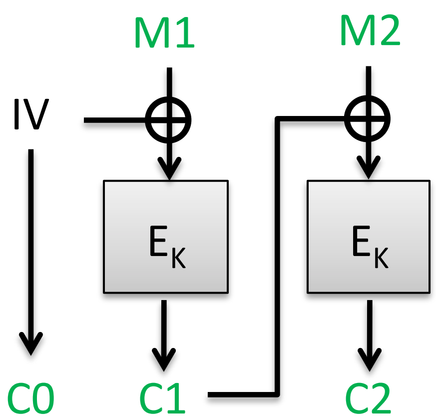
- Edward Snowden, May 2013

Some failures of symmetric encryption: 1970s – today



Example 1: primitive failure

- DES with 56-bit keys
- RC4 plaintext recovery attacks [Paterson, Poettering, Schuldt 14]



Example 2: active attack failures

- CBC mode [Vaudenay 02, …] [Rizzo, Duong 11]
- MAC-then-Encrypt [Alfarden, Paterson 13] [Paterson, R., Shrimpton 12] [Degabriele, Paterson 10]

Early release of plaintext

Power, timing, access-driven side channel attacks

Backdoors in PRNGs

Solving all ***those*** problems won't directly help *censorship victims* and *LastPass users*

Deep packet inspection systems can block protocols

→ Ciphertexts don't "look like" benign traffic to network monitors

LastPass uses password-based encryption that can be cracked

→ Decryption reveals when wrong key is used

Traditional approach: punt on such problems to systems security

Our approach: new symmetric encryption primitives

Today's talk

- Part 1: Format-transforming encryption

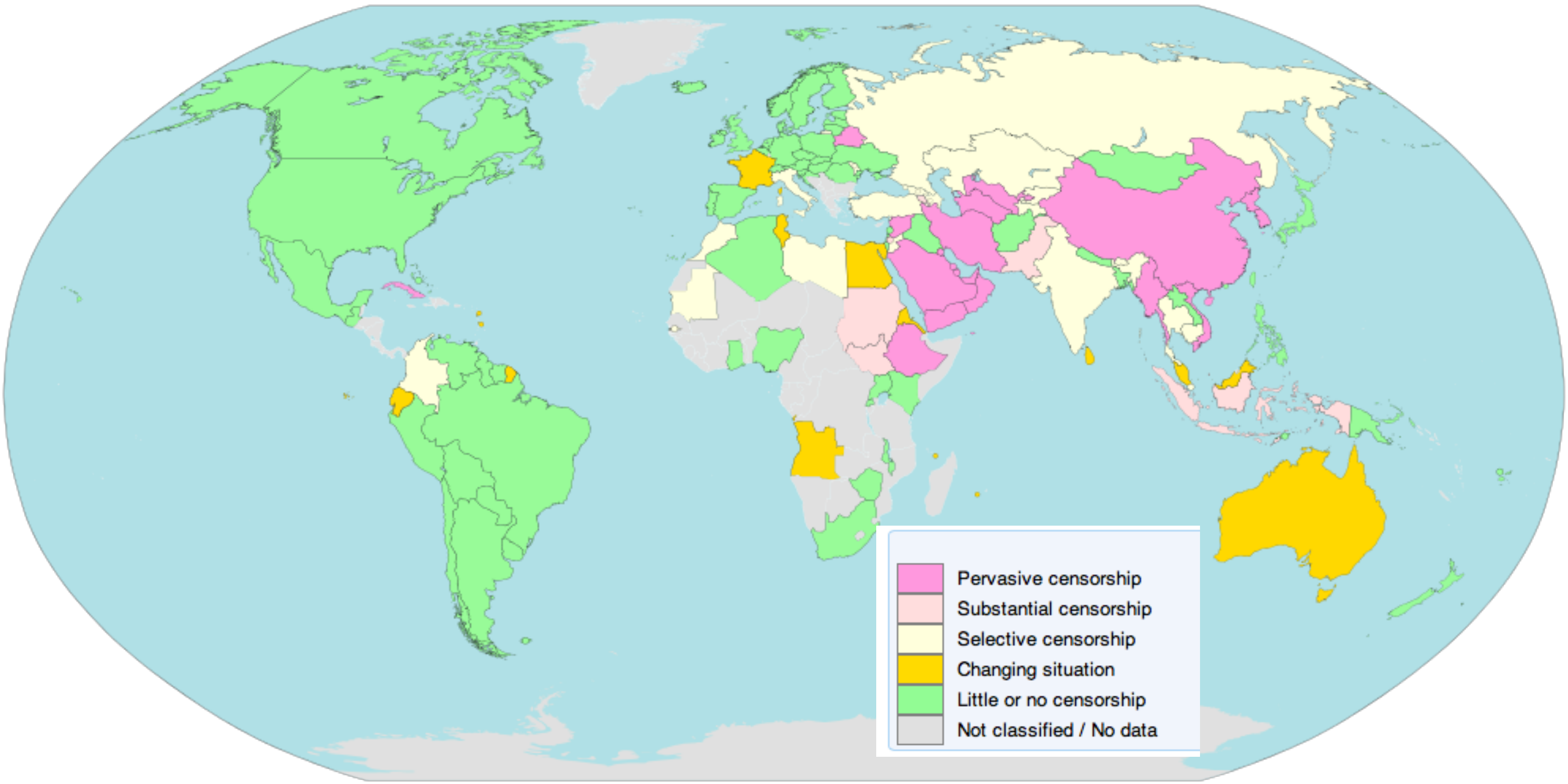
  [Dyer, Coull, R., Shrimpton – CCS 2013]

- Part 2: Honey encryption

  [Juels, R. – Eurocrypt 2014]

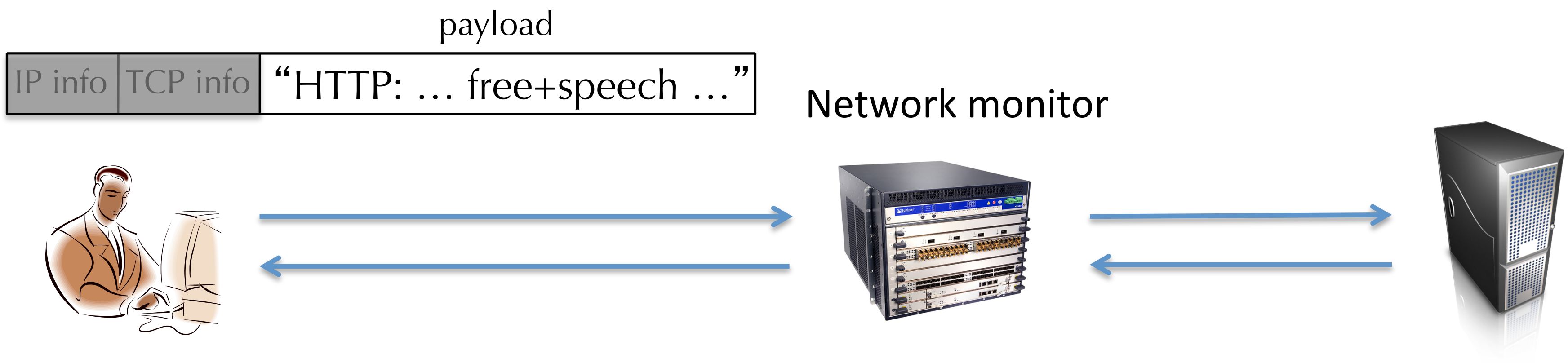
Current Estimates of Internet Censorship

OpenNet Initiative (ONI),
Reporters Without Borders
(via wikipedia; updated Jan 6, 2014)



Magenta-colored countries are **"internet black holes"**: have heavy censorship of political, social, and news sites, internet tools, etc.

Packet inspection and existing countermeasures

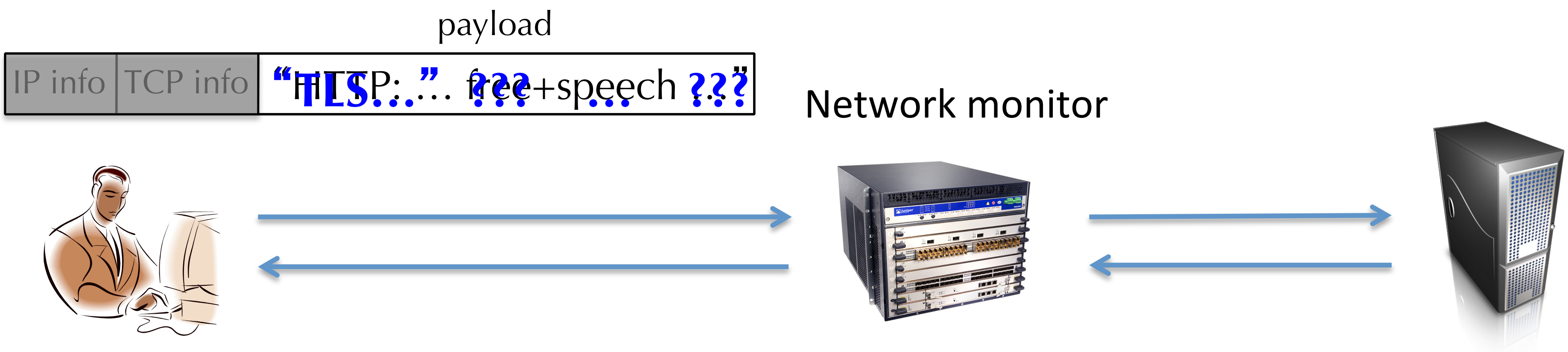


A packet can tell you:

- source address
- destination address/port
- application-level protocols
- keywords in payloads
- …

Use a proxy service, e.g. Tor

Packet inspection and existing countermeasures



A packet can tell you:

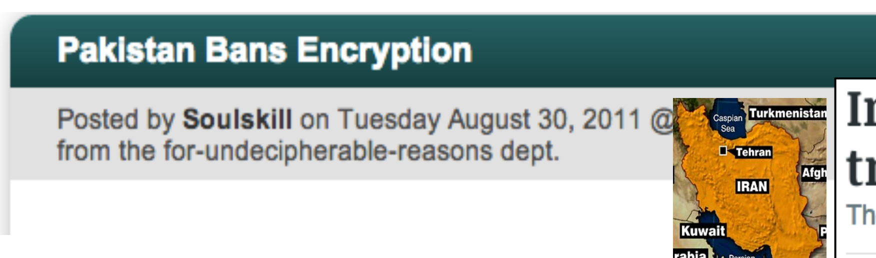
- source address
- destination address/port
- application-level protocols
- keywords in payloads
- …

Making payload information unhelpful is a new challenge

Why not just use standard encryption tools?

Hides the protocol/content inside the encrypted tunnel…

But use of the encryption protocol is still visible.



Pakistan Bans Encryption

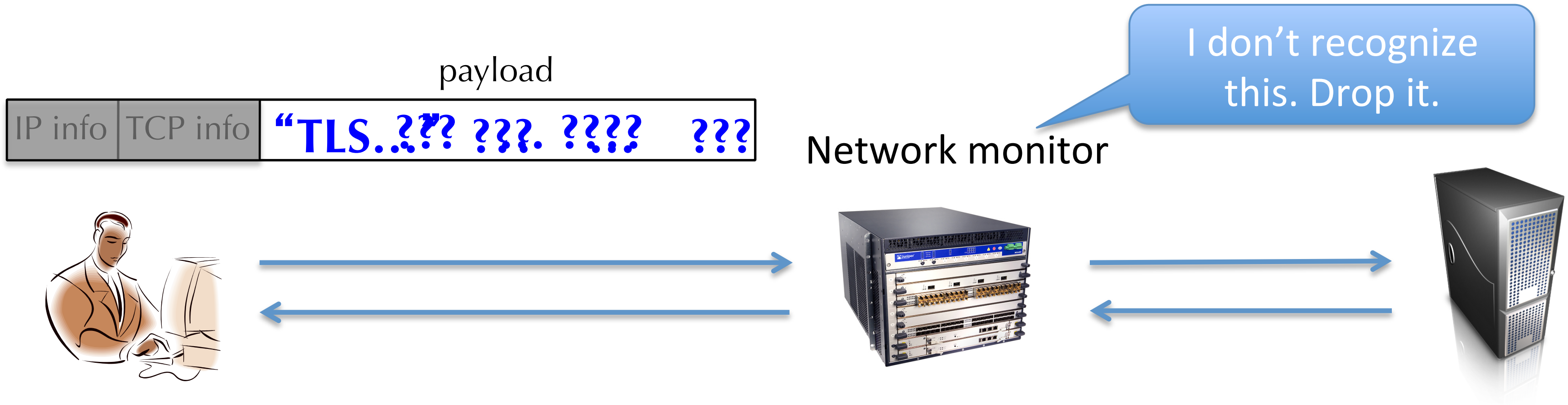
Posted by **Soulskill** on Tuesday August 30, 2011 @
from the for-undecipherable-reasons dept.

Iran reportedly blocking encrypted Internet traffic

The Iranian government is reportedly blocking access to websites that use the …

by **Jon Brodkin** - Feb 10 2012, 9:44pm IST

Packet inspection and existing countermeasures



Making payload information unhelpful is a new challenge

A packet can tell you:

- source address
- destination address/port
- application-level protocols
- keywords in payloads
- …

Why not make *all packet contents random*?

Used by obfsproxy for Tor

What happens if DPI allows only whitelisted protocols?

Some previous efforts in DPI Circumvention

Stegotorus [Weinberg et al., 2012],

SkypeMorph [Moghaddam et al. 2012],

FreeWave [Houmansadr et al., 2013], etc.

These represent nice steps in the right direction, but

1. **Poor performance:** 16-256 Kbps reported (best case)

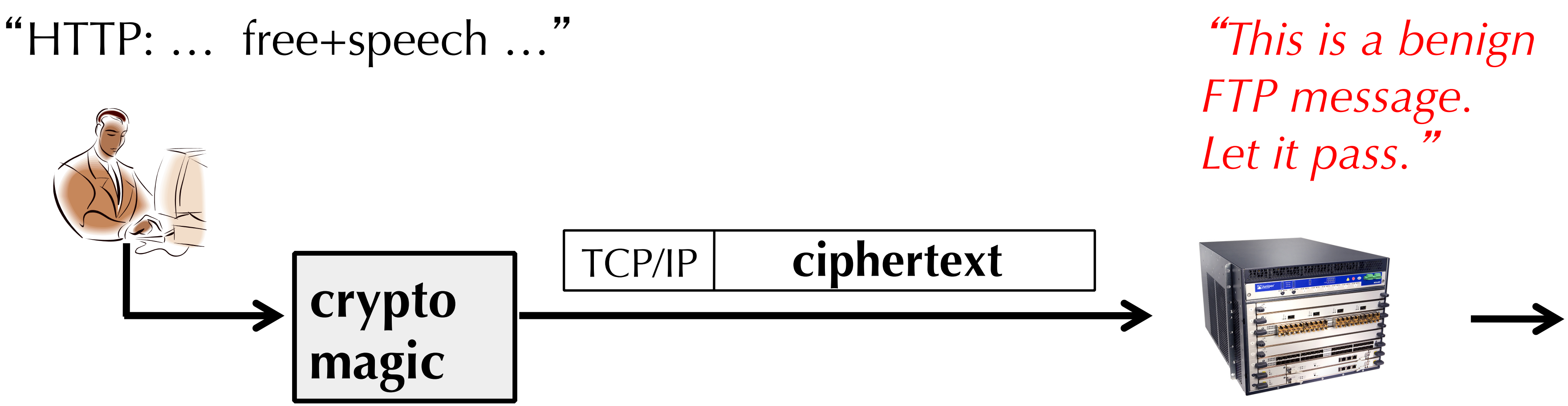
2. **Inflexible:** not quickly adaptable to changes in DPI rules.

   e.g. what if you're using SkypeMorph,
   and Skype becomes blocked? (Ethiopia 2013)

3. **Not empirically validated:** do they work against real DPI?

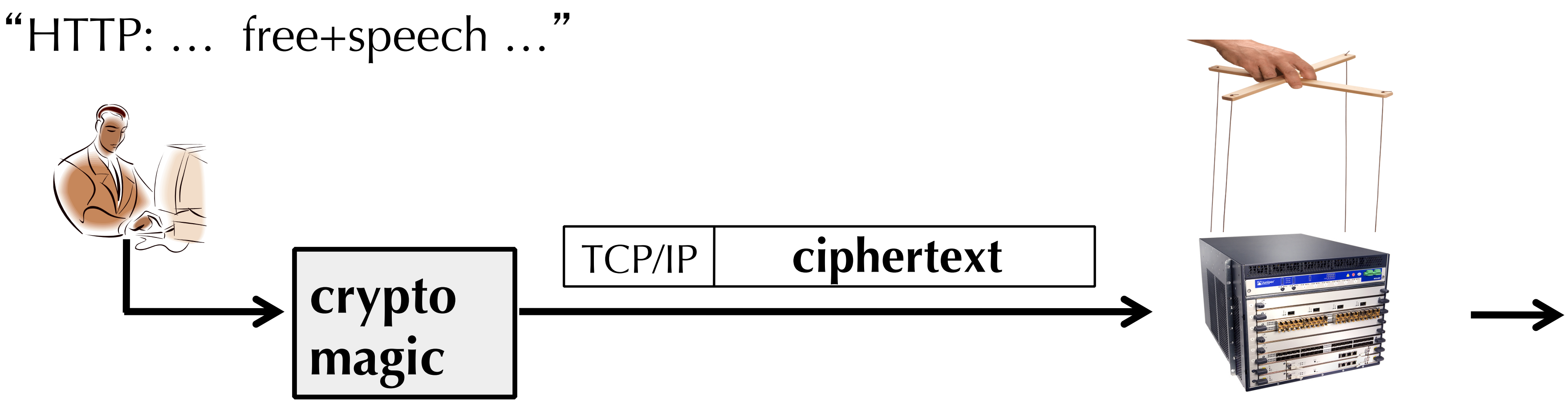
Our goal: cause real DPI systems to reliably misclassify our traffic

for example: HTTP misclassified as FTP



(and in a way that is flexible and has good throughput/low latency…)

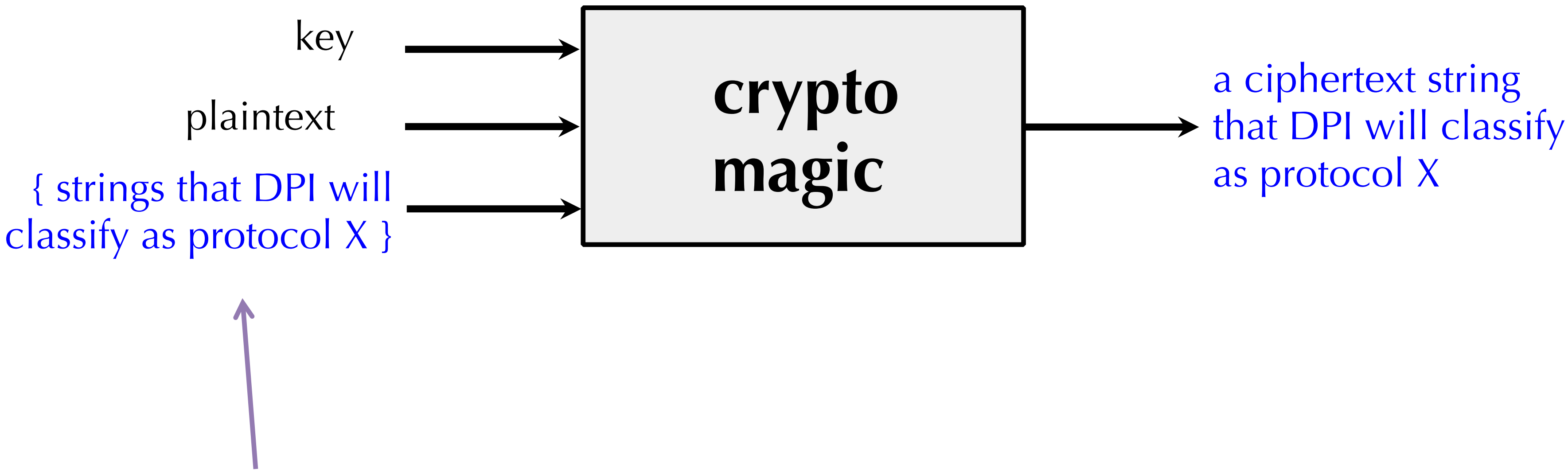
Our goal: cause real DPI systems to reliably misclassify our traffic **as whatever protocol we want.**



(and in a way that is flexible and has good throughput/low latency…)

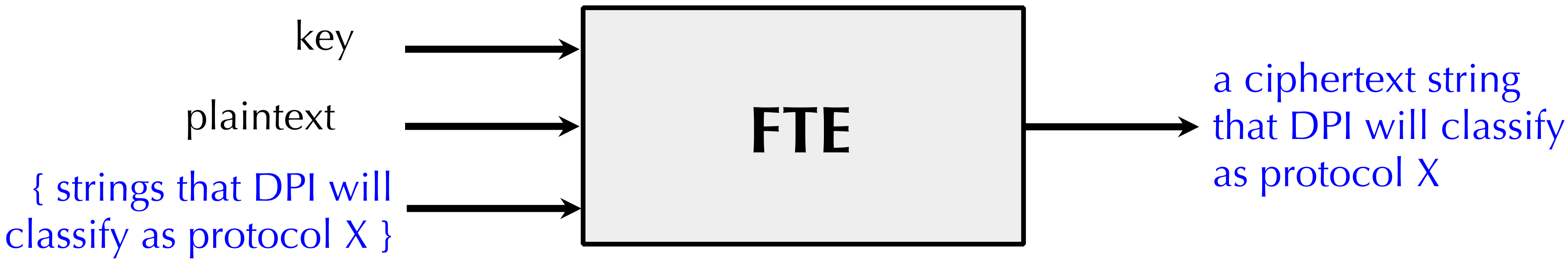
We took inspiration from Format-Preserving Encryption

[Bellare et al., 2009]



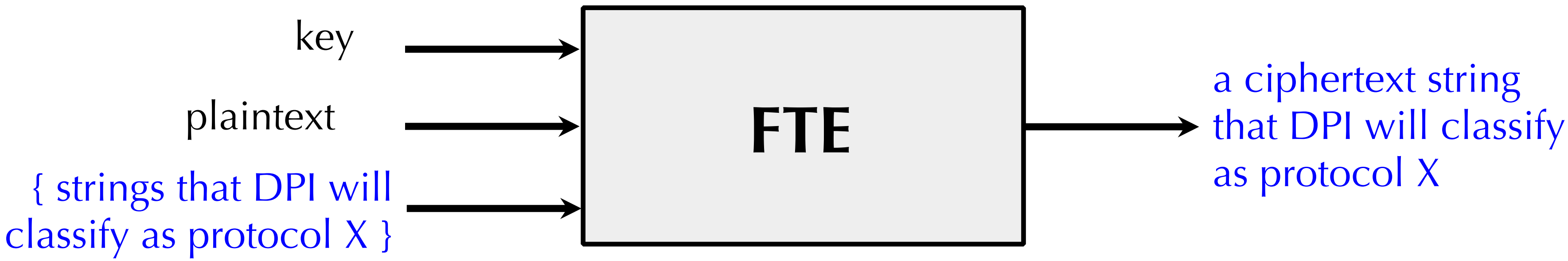
The desired ciphertext “format”

Format-*Transforming* Encryption



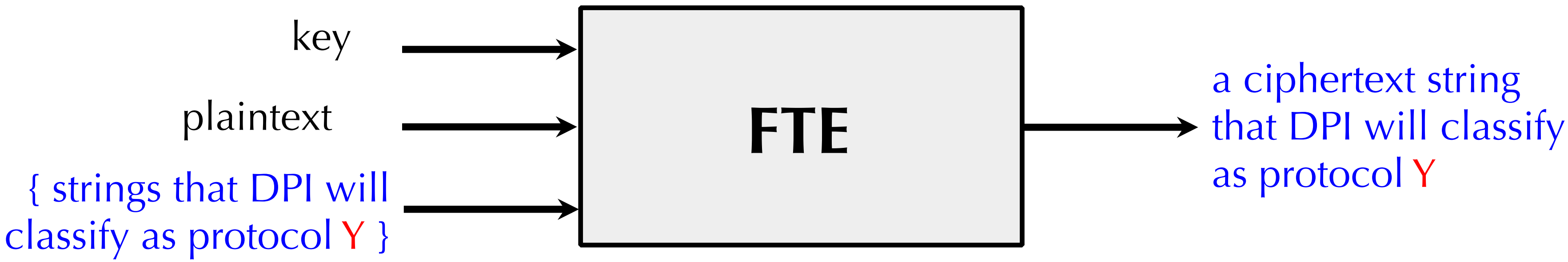
Like traditional encryption, with the extra operational requirement that ciphertexts fall within the format.

Ciphertext flexibility is built into the FTE syntax



Adapting to new DPI rules or different protocols requires changing only the format

Ciphertext flexibility is built into the FTE syntax



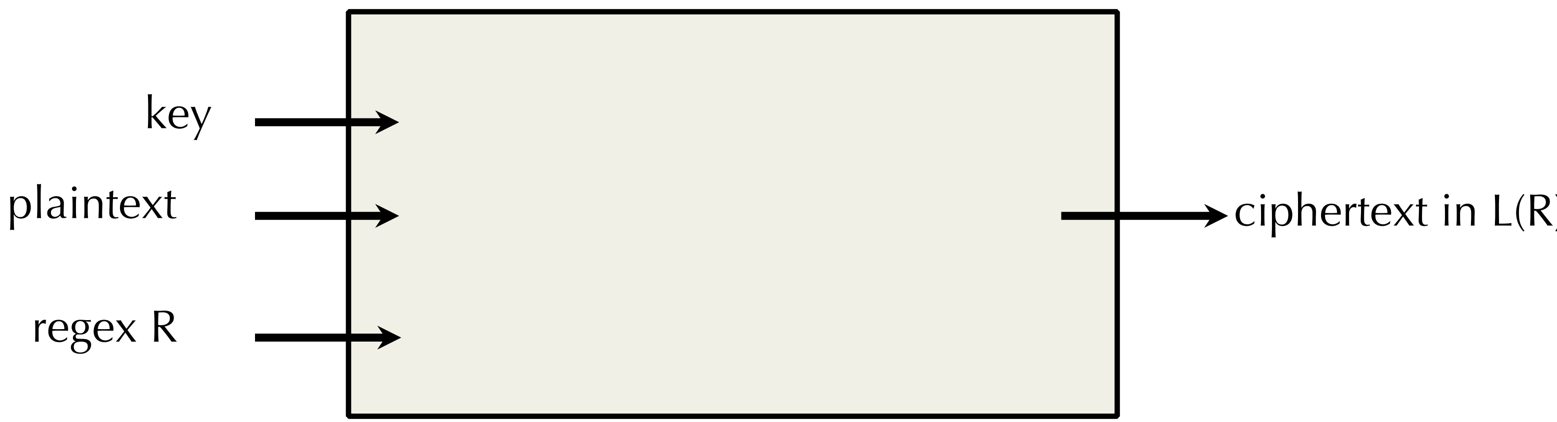
Adapting to new DPI rules or different protocols requires changing only the format

Surveying modern DPI systems

System	Protocol classification uses	Costs
AppID	**Regular expressions**	Free
L7-filter	**Regular expressions**	Free
Yaf	**Regular expressions** (sometimes hierarchical)	Free
Bro	Simple **regular expression** triage, then additional parsing and heuristics	Free
nProbe	Parsing and heuristics (many of them "**regular**")	~300 euros
Proprietary	**???**	~10,000 USD

Can we build FTE schemes that support formats defined by regexes?

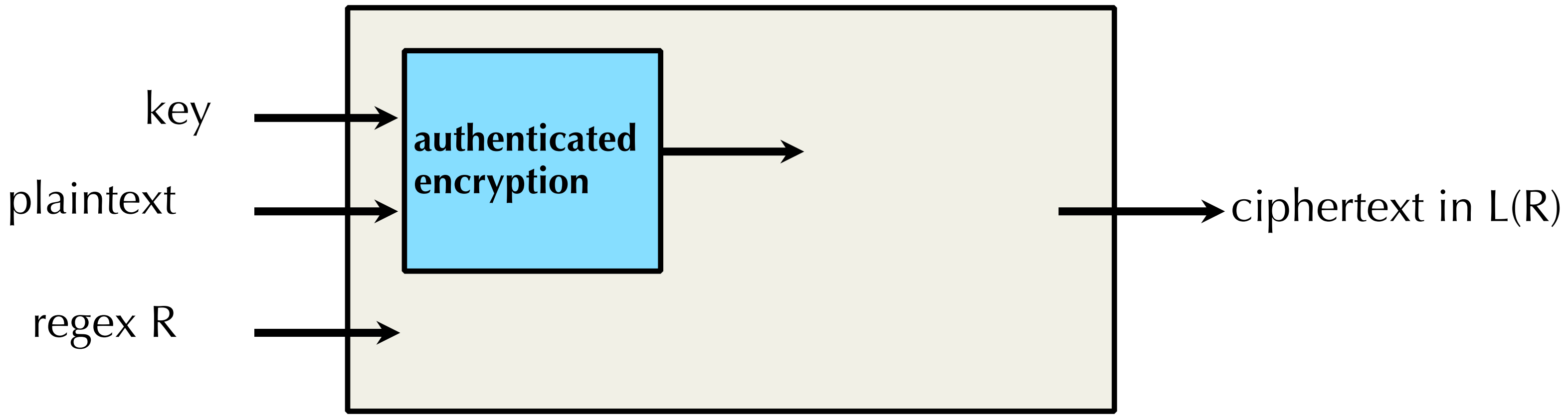
Realizing regex-based FTE



How should we realize regex-based FTE?

We want:
- Cryptographic protection for the plaintext
- Ciphertexts in L(R)

Realizing regex-based FTE

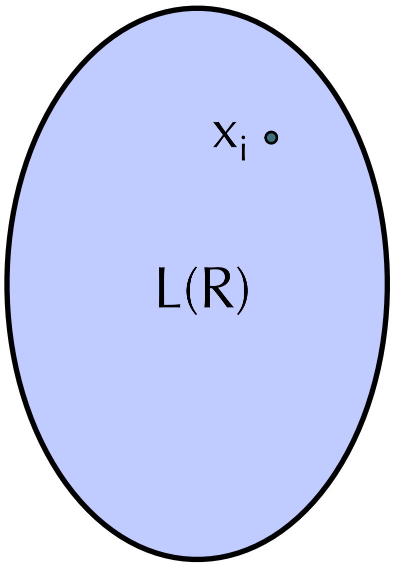


How should we realize regex-based FTE?

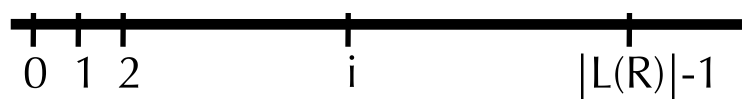
We want: Cryptographic protection for the plaintext
Ciphertexts in L(R)

Ranking a Regular Language

[Goldberg, Sipser '85]
[Bellare et al. '09]



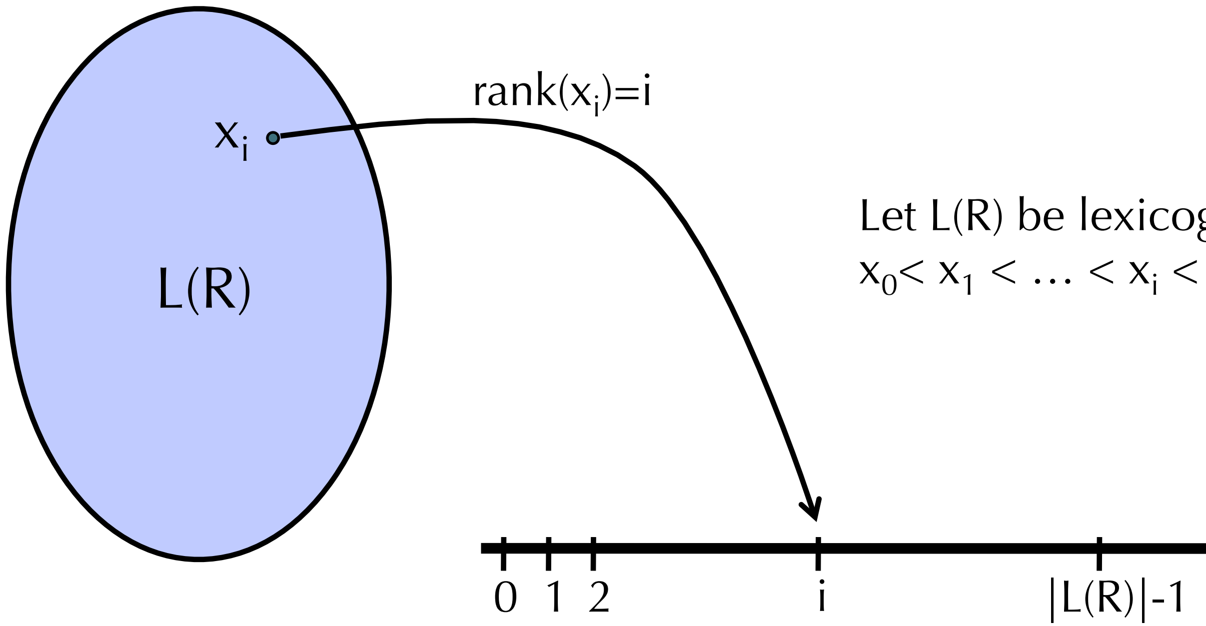
Let L(R) be lexicographically ordered
$x_0 < x_1 < \ldots < x_i < \ldots < x_{|L(R)-1|}$



Given a **DFA** (deterministic finite automaton) for L(R),
there are efficient algorithms

Ranking a Regular Language

[Goldberg, Sipser '85]
[Bellare et al. '09]



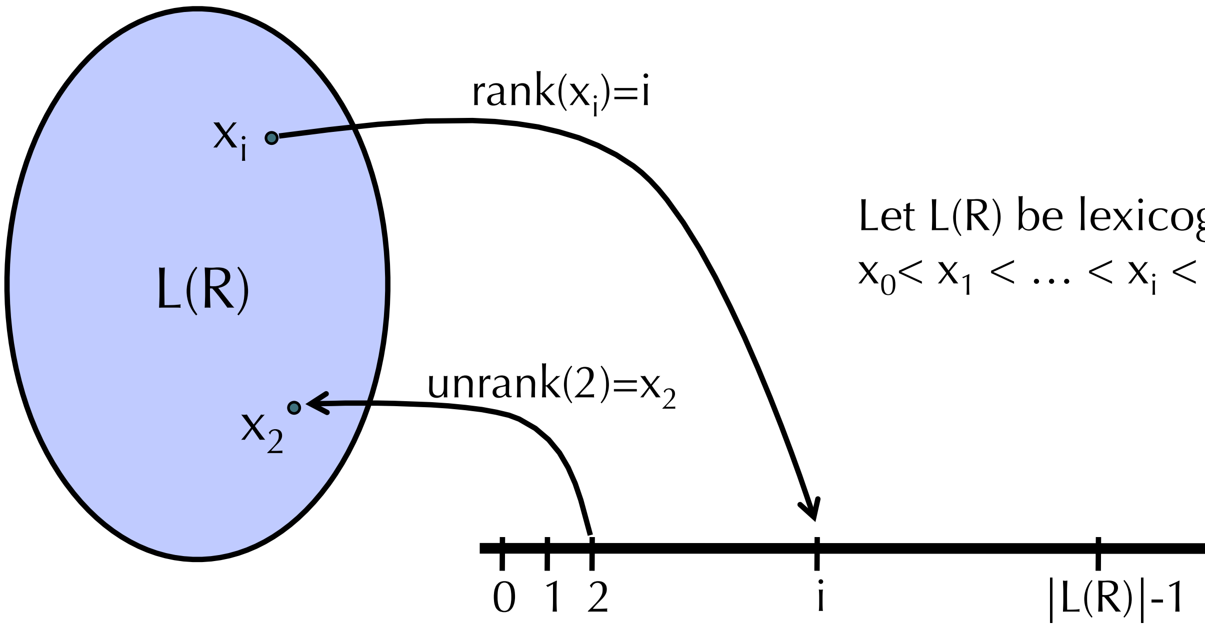
Let L(R) be lexicographically ordered
$x_0 < x_1 < \ldots < x_i < \ldots < x_{|L(R)-1|}$

Given a **DFA** (deterministic finite automaton) for L(R),
there are efficient algorithms

$$\text{rank: } L(R) \longrightarrow \{0,1,\ldots,|L(R)|-1\}$$

Ranking a Regular Language

[Goldberg, Sipser '85]
[Bellare et al. '09]



Let L(R) be lexicographically ordered
$x_0 < x_1 < \ldots < x_i < \ldots < x_{|L(R)-1|}$

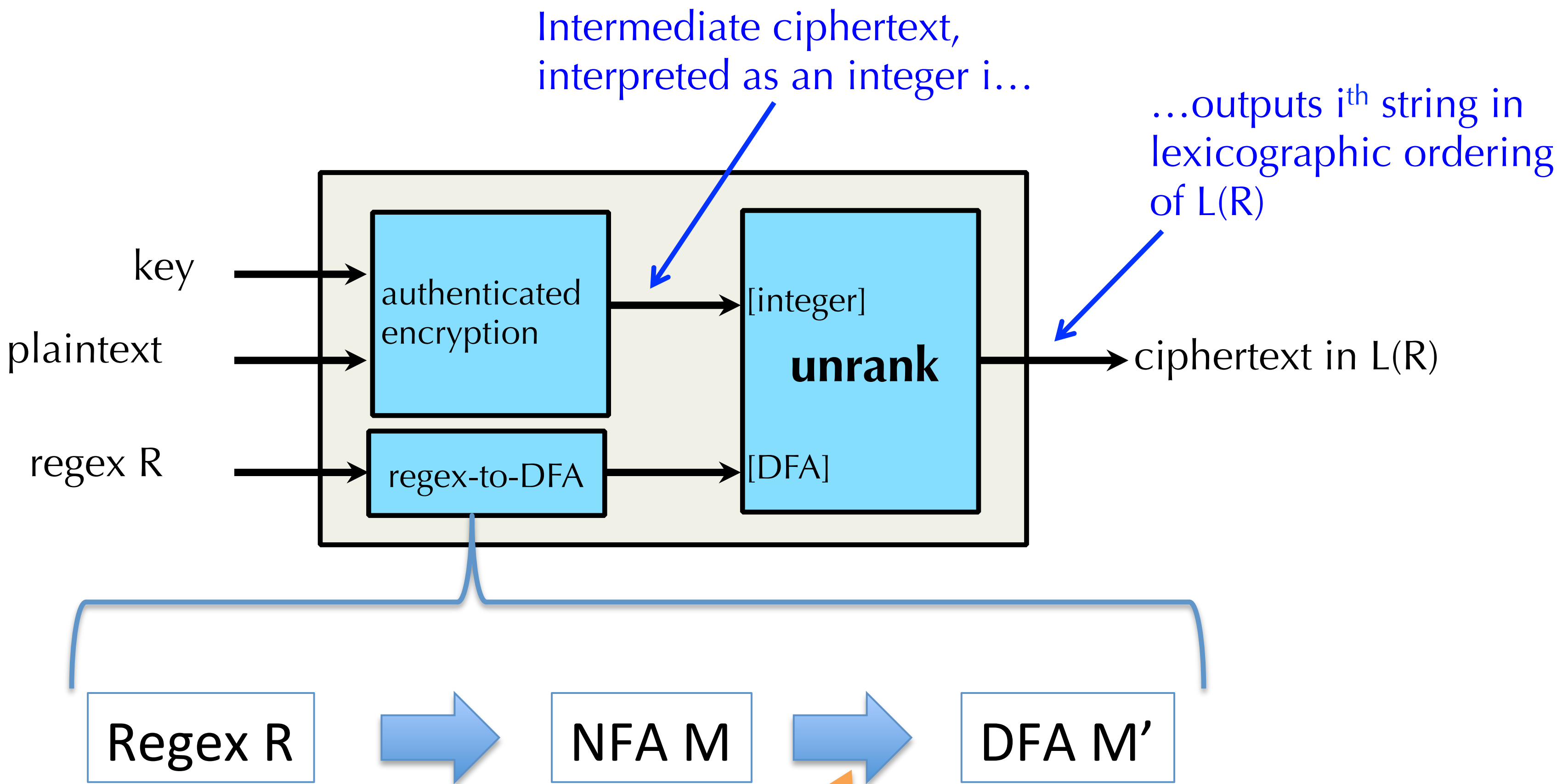
Given a **DFA** (deterministic finite automaton) for L(R),
there are efficient algorithms

rank: L(R) $\longrightarrow$ {0,1,…,|L(R)|-1}
unrank: {0,1,…,|L(R)|-1} $\longrightarrow$ L(R)

such that rank(unrank(i)) = i
and unrank(rank(x_i)) = x_i

With precomputed tables, rank, unrank are *O*(n)

Realizing regex-based FTE

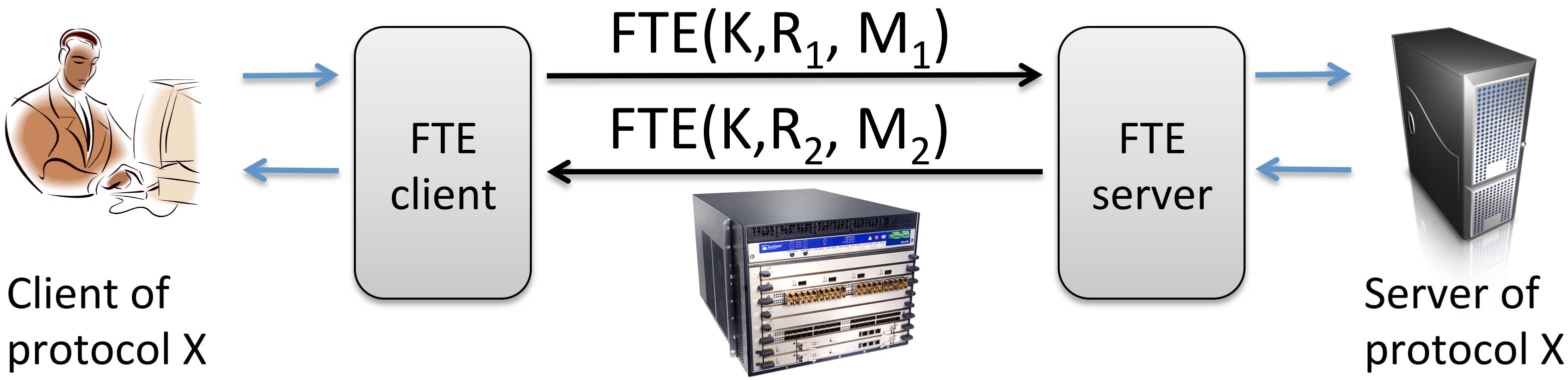


Exponential blow-up in worst case. Regexes we needed avoid this.

FTE using NFAs directly

[Luchaup, Dyer, Jha, R., Shrimpton –
In submission 2014]

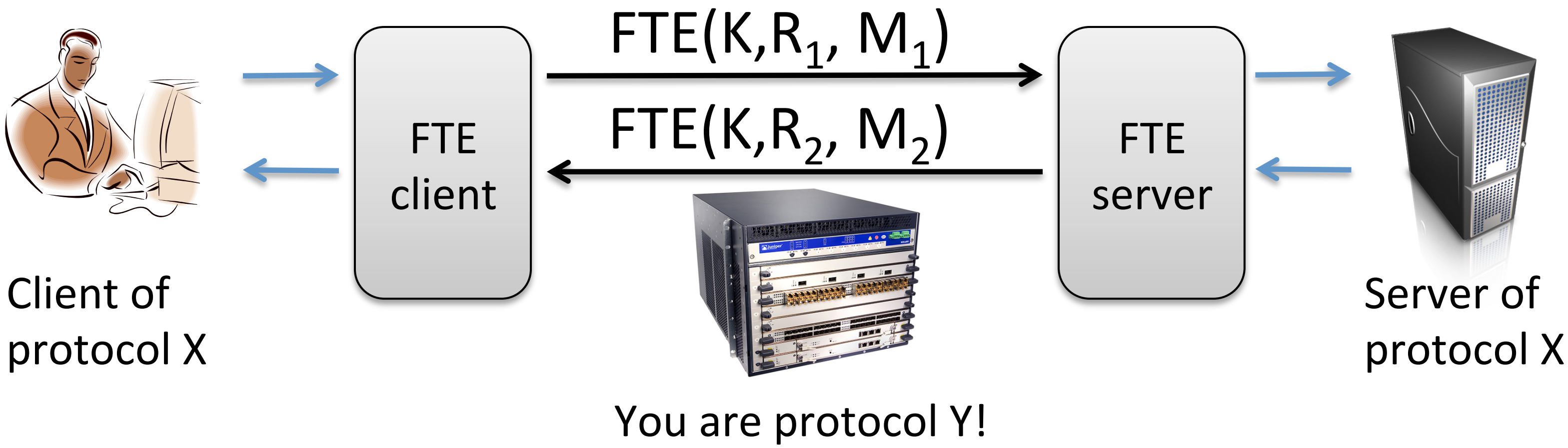
We built a complete FTE record layer and proxy system



Involved significant engineering effort.

Paper has more details or ask Kevin Dyer

We built a complete FTE record layer and proxy system



Want to trick DPI into thinking we're protocol Y != X

Where do we get R_1 and R_2?

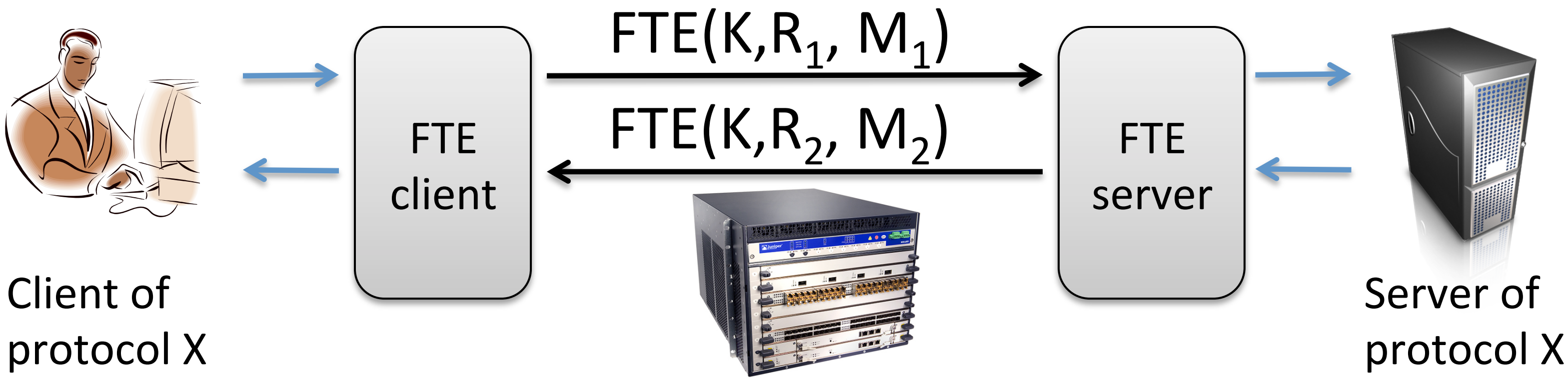
(1) Get from DPI themselves

(2) Easy to manually craft

(3) Learn from traffic samples

We built regexes for variety of "cover" protocols: Y = HTTP, SSH, SMB, SIP, RTSP

Evaluating FTE

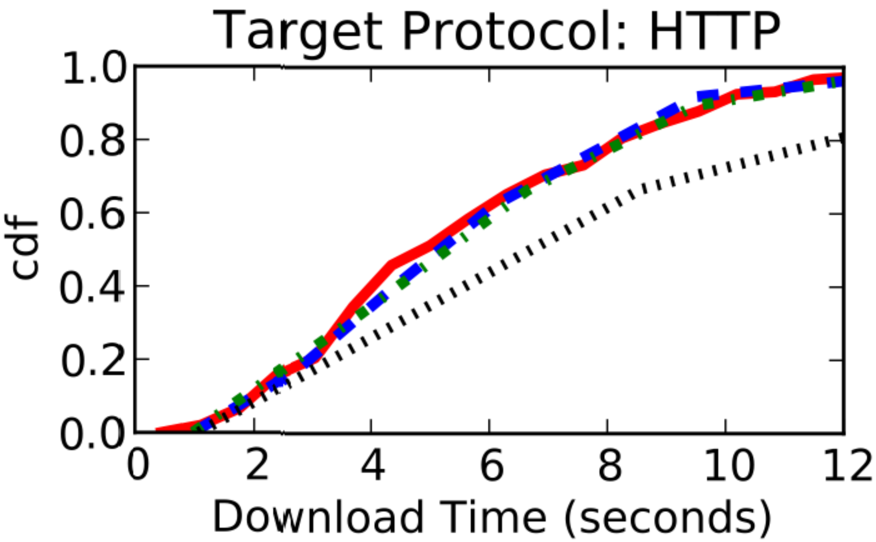


Tests with gets on Alexa Top 50 sites (X = mix of HTTPS/HTTP)

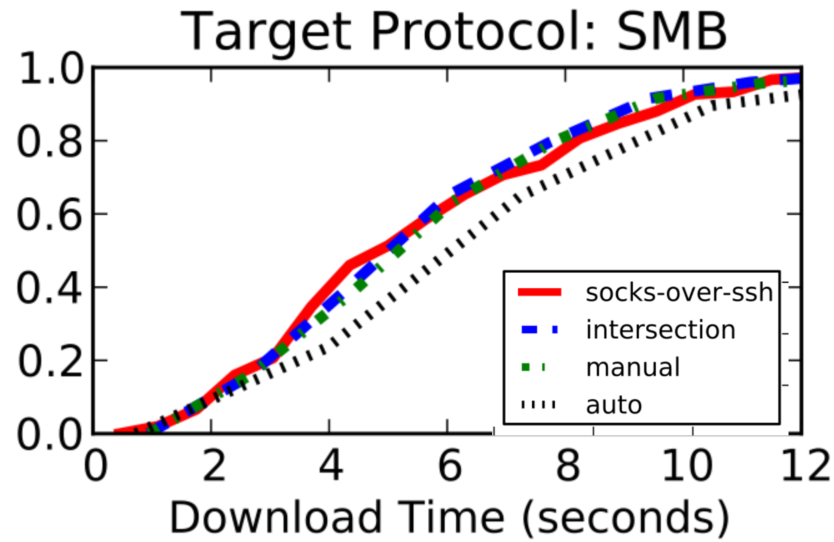
R_1 R_2 set to HTTP, SSH, SMB, and more. When do we trick DPI ?

System	DPI-derived regex's	Manual regex's	Learned regex's
AppID	Always	Always	Always
L7-filter	Always	Always	Always
Yaf	Always	Always	Always
Bro	Sometimes	Always	Always
nProbe	Never	Always	Almost always
Proprietary	Always	Always	Always

Web-browsing performance



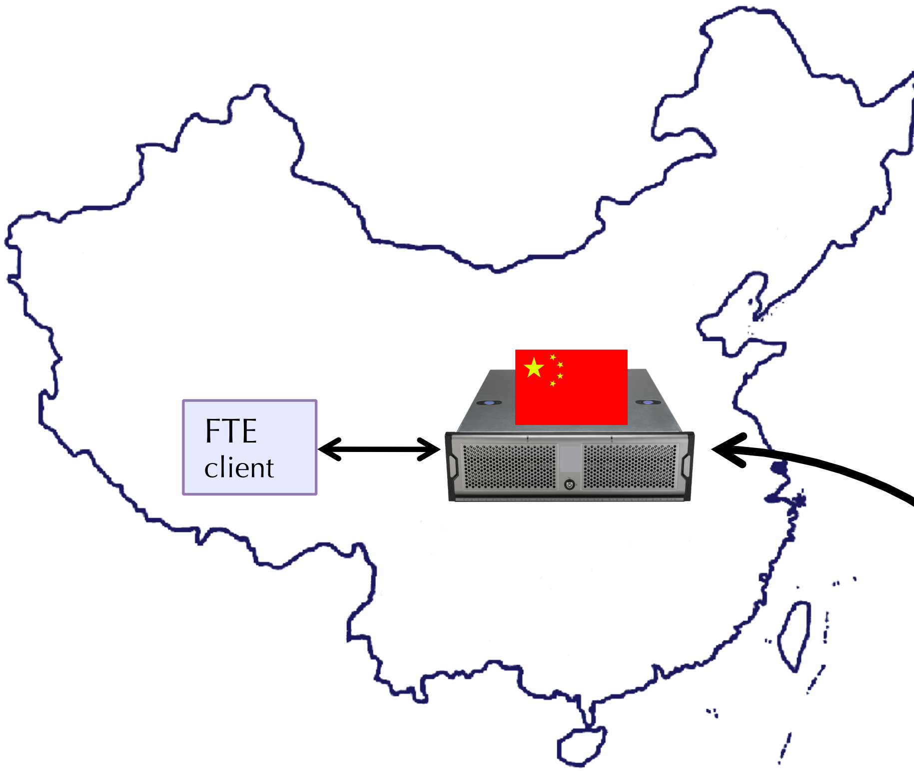
Top 50 Alexa websites



Top 50 Alexa websites

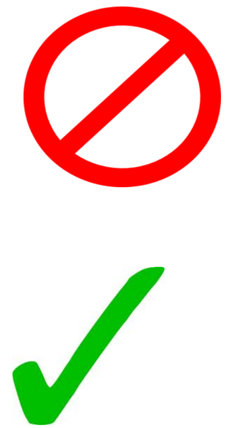
Punchline: FTE or SSH tunnel result in the same user web-browsing experience

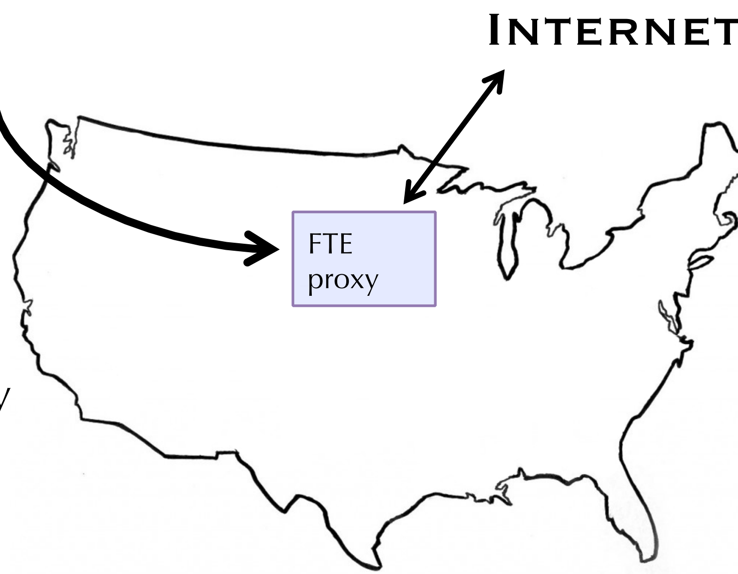
A field test…



Without FTE tunnel, we tried Facebook, YouTube, Tor website, banned search queries…

With FTE tunnel, we tried Facebook, YouTube, Tor website, banned search queries…





Used FTE to download Tor bundle:

Tor without FTE: “active blacklisting” attack on proxy
Tor through FTE: no problems

Ran various tests every 5 minutes for one month,
no sign of detection in logs. (We shut it down after that.)

FTE is open source,
runs on multiple platforms/OS, and
fully integrated into Tor.

Undergoing beta tests for use
in Tor bundle clients

Lantern also incorporating FTE into their
anti-censorship tool

http://fteproxy.org

Today's talk

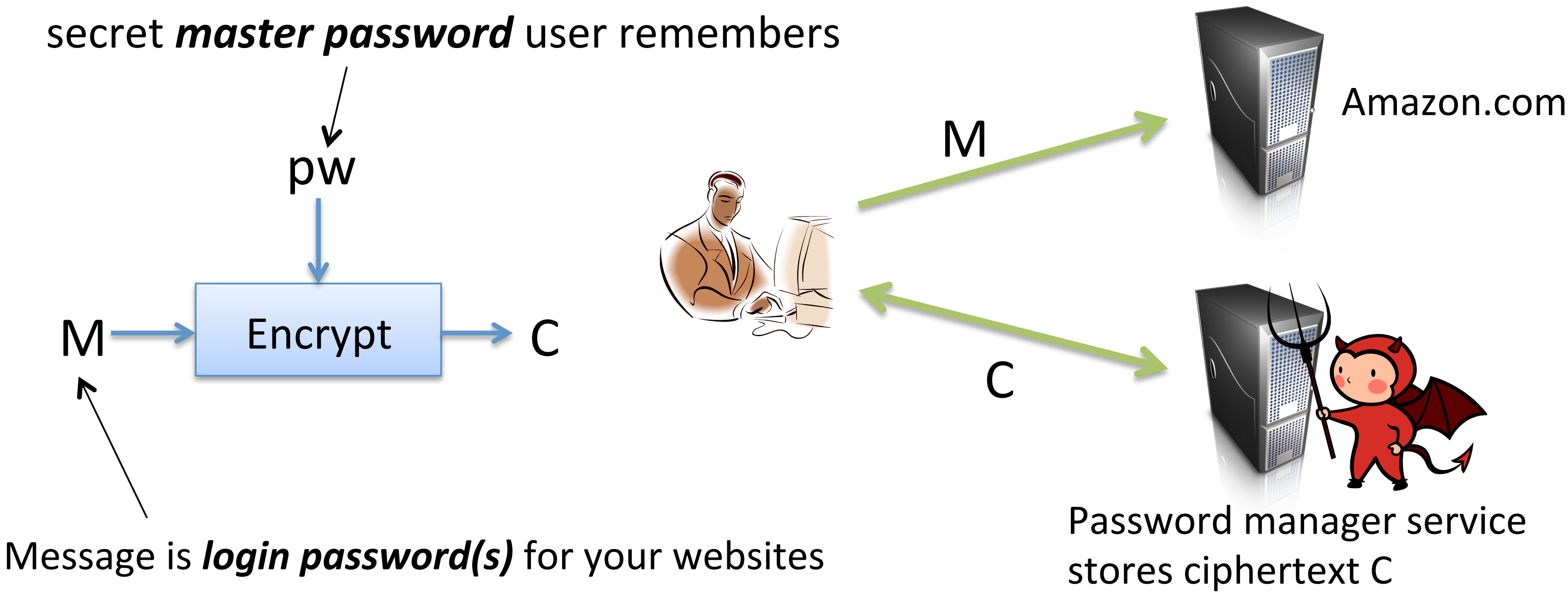
- Part 1: Format-transforming encryption

  [Dyer, Coull, R., Shrimpton – CCS 2013]

- Part 2: Honey encryption

  [Juels, R. – Eurocrypt 2014]

Password-based encryption example



LastPass CEO reveals details on security breach

by Lance Whitney | May 6, 2011 10:19 AM PDT

Internet users ditch "password" as password, upgrade to "123456"

Contest for most commonly used terrible password has a new champion.

by **Jon Brodkin** - Jan 20 2014, 4:00pm GMT

Rank	Password	Change from 2012
1	123456	*Up 1*
2	password	*Down 1*
3	12345678	*Unchanged*
4	qwerty	*Up 1*
5	abc123	*Down 1*
6	123456789	*New*
7	111111	*Up 2*
8	1234567	*Up 5*
9	iloveyou	*Up 2*
10	adobe123	*New*

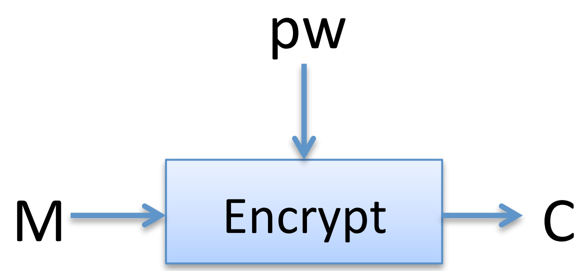
Source:
Splash Data
http://splashdata.com/press/worstpasswords2013.htm

[Bonneau 2012]
69 million Yahoo! Passwords
1.1% of users pick same password

People choose weak passwords

Brute-force attacks against ciphertext

Master password pw drawn from set $\{pw_1, pw_2, \ldots, pw_q\}$
(e.g., $q = \sim 10^6$)



<u>Brute force attack given C:</u>

M_1 <- Decrypt(pw_1,C) → abufdsjkl!feqfdsj

M_2 <- Decrypt(pw_2,C) → hgjk!alc&ewj*ofw

M_3 <- Decrypt(pw_3,C) → password123

… → …

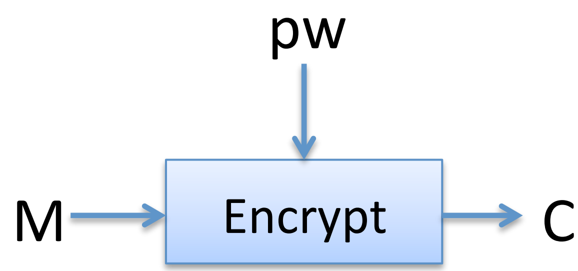
M_q <- Decrypt(pw_q,C) → tyei01agjzfjfdajsal

Password-based encryption

- PKCS#5 standard:
  - Slow down decryption by lots of hashing and use salts
  - Provably works … [Bellare, R., Tessaro – Crypto 12]
  - … but only slows down previous attack by constant factor

Embedding decoys into encryption?

Master password pw drawn from set $\{pw_1, pw_2, \ldots, pw_q\}$ (e.g., $q = \sim 10^6$)



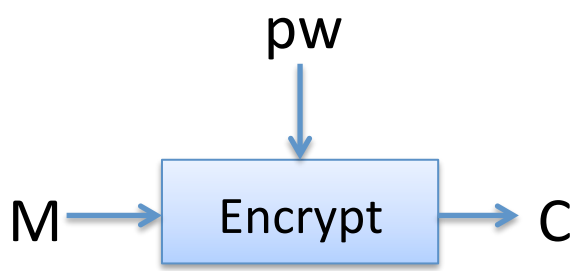
What if we could build encryption so that:

<u>Brute force attack given C:</u>

M_1 <- Decrypt(pw_1,C)	abufdsjkl!feqfdsj
M_2 <- Decrypt(pw_2,C)	hgjk!alc&ewj*ofw
M_3 <- Decrypt(pw_3,C)	password123
…	…
M_q <- Decrypt(pw_q,C)	tyei01agjzfjfdajsal

Embedding decoys into encryption?

Master password pw drawn from set $\{pw_1, pw_2, \ldots, pw_q\}$
(e.g., $q = {\sim}10^6$)



What if we could build encryption so that:

<u>Brute force attack given C:</u>

M_1 <- Decrypt(pw_1,C)
M_2 <- Decrypt(pw_2,C)
M_3 <- Decrypt(pw_3,C)
…
C)

??? →

123456789
11111
password123
…
adobe123

Attacker would have to try logging in with decoy passwords

Decoys in computer security

- Decoys, fake objects that look real, are a time-honored counterintelligence tool.
- In computer security, we have "honey objects":
  – Honeypots [S02]
  – Honeytokens, honey accounts
  – Decoy documents [BHKS09] (many others by Keromytis, Stolfo, et al.)
  – Honeywords for password hashing [JR13]

Password vaults are just one kind of message

- RSA secret keys
  - Uniform bit strings as secret exponents [HK99]
- Cookies, other bearer tokens, other authentication values
- Non-authenticaton related?
  - English language text

Honey encryption

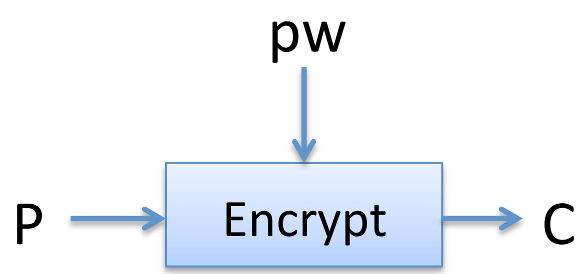
[Juels, R. – Eurocrypt 2014]

- Same API as password-based encryption schemes
  - Arrange to be secure in sense of [BRT12] (keep salting and hash chains)
- Use special encodings to ensure that decrypting ciphertext with ***wrong*** key yields fresh sample from designer's estimate of message distribution
- Good encoding:
  attacker provably can't pick out right message

Honey encryption for prime numbers

Useful to store secret keys for some authentication systems (RSA) [HK99]

n=1024-bit prime number chosen uniformly



Attacker can run primality tests to see which is prime.
Each M_i is prime w/ probability 1 / 1024

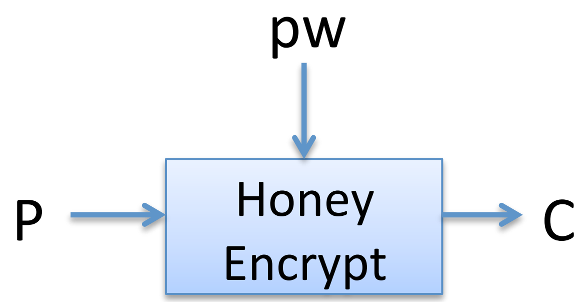
<u>Brute force attack given C:</u>

M_1 <- Decrypt(pw_1,C)	100
M_2 <- Decrypt(pw_2,C)	321849
M_3 <- Decrypt(pw_3,C)	9883
…	…
M_q <- Decrypt(pw_q,C)	16

Honey encryption for prime numbers

Useful to store secret keys for some authentication systems (RSA) [HK99]

n=1024-bit prime number chosen uniformly



All outputs of decryption are uniformly distributed prime numbers!

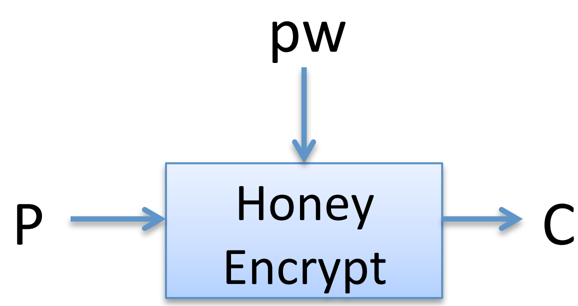
<u>Brute force attack given C:</u>

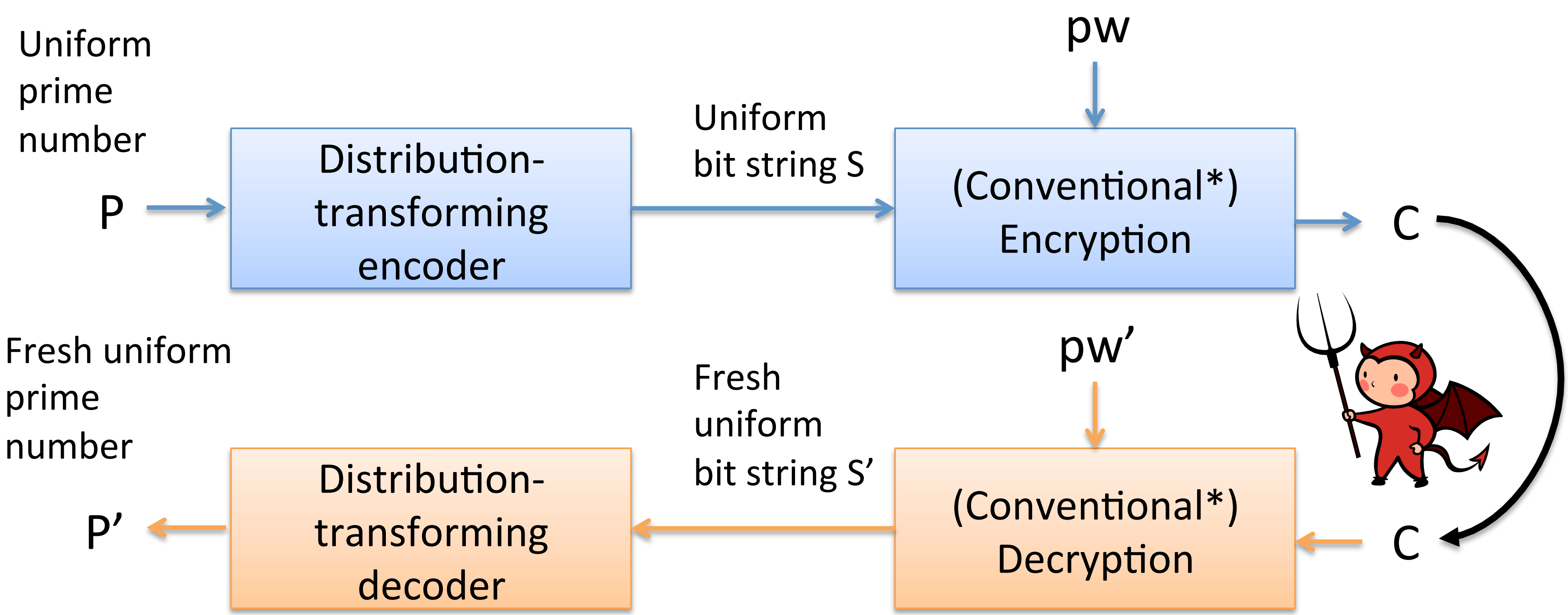
M_1 <- Decrypt(pw_1,C)		102953
M_2 <- Decrypt(pw_2,C)	???	56431
M_3 <- Decrypt(pw_3,C)		9883
…		…
M_q <- Decrypt(pw_q,C)		26171

Honey encryption for prime numbers

Useful to store secret keys for some authentication systems (RSA) [HK99]

n=1024-bit prime number chosen uniformly

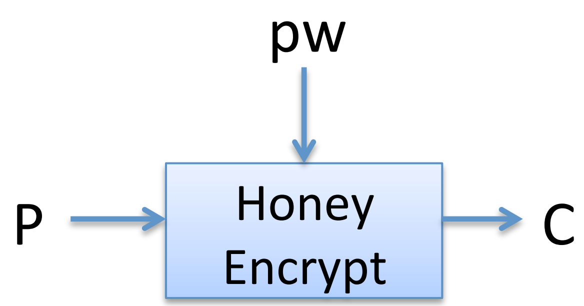


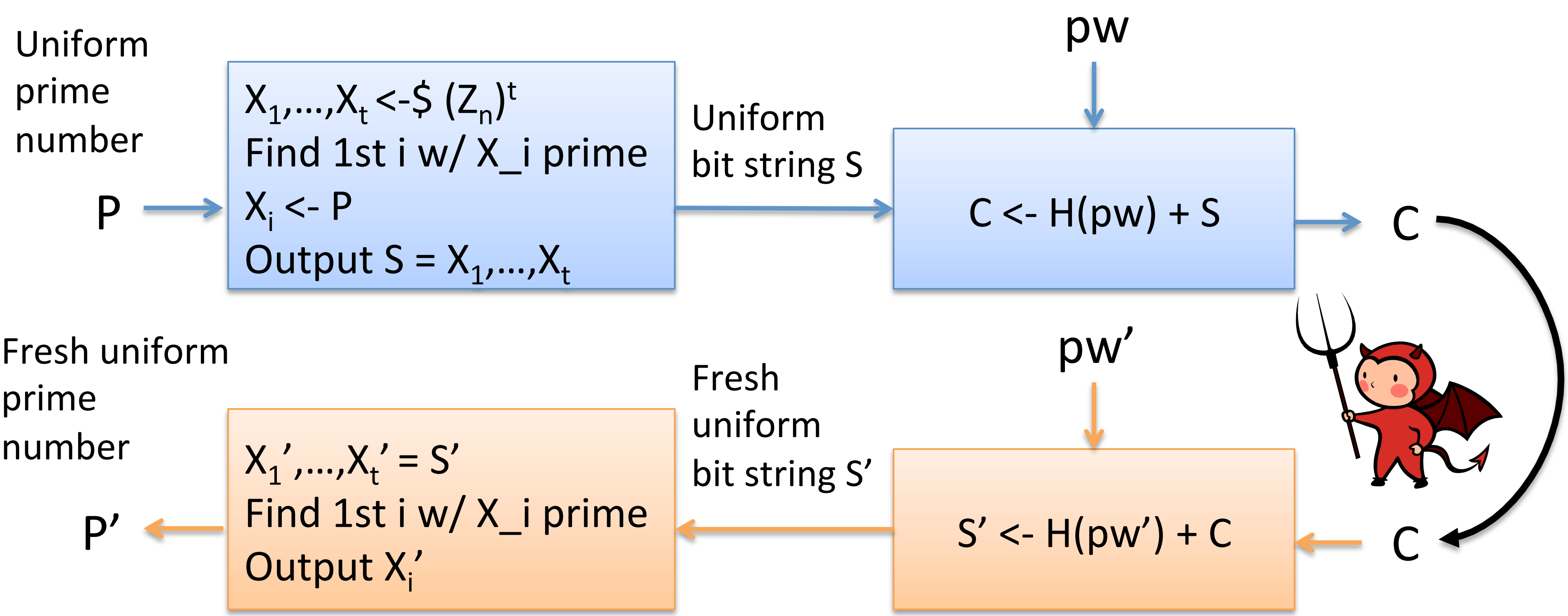


Honey encryption for prime numbers

Useful to store secret keys for some authentication systems (RSA) [HK99]

n=1024-bit prime number chosen uniformly

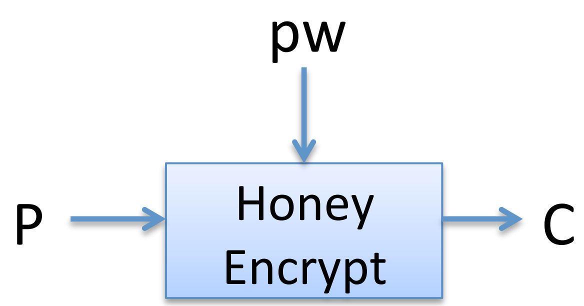




Honey encryption for prime numbers

Useful to store secret keys for some authentication systems (RSA) [HK99]

n=1024-bit prime number chosen uniformly



Thm (informal). No attacker A can recover correct message with probability better than ~ 1 / q

Security bound is optimal!

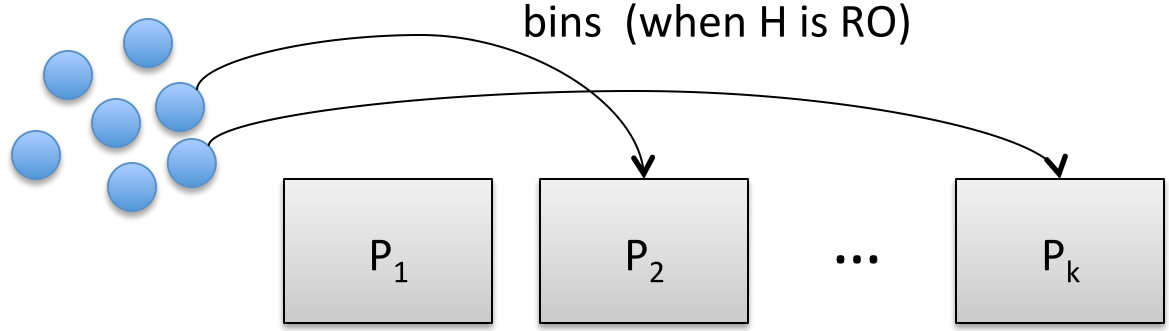
Intuition for proof

<u>MR Game:</u>
P <-$ GenPrime()
pw <-$ GenKey()
S <-$ Encode(P)
C <- H(pw) + S
P' <- A^H(C)
Ret (P=P')

Can view experiment as a ***balls-and-bins game***

Balls are keys
(Equal weight 1/q for uniform distribution)

Balls thrown independently into bins (when H is RO)



Bins are possible messages.
Equal-sized if decoded primes uniform
(t must be large enough)

Adversary's advantage maximized by picking bin at end of game with most balls

Expected maximum load E[L] is expected weight of maximally weighted bin

Well-studied for some settings

In this case: if $q^2 << k$ then
E[L] = 1/q + negl

We give broader analysis framework

- Keys (passwords) are not uniformly chosen
  - Weights of balls differ (use theory of majorization)
- Message spaces not always uniform
  - E.g.: non-uniform primes (OpenSSL), credit card # w/ pin, website passwords
  - Bin sizes differ
- See paper for more details

Honey encryption: the future

- In paper only give DTEs for some message types
  - Uniform and non-uniform prime numbers
  - Credit-card numbers (w/ PINs)
- Want to build ones for messages being
  - Passwords (to help out poor Lastpass)
    - Already have some working prototypes
  - Others?
- Operational considerations
  - Typo safety
  - Detection of online attacks
  - Further deployment scenarios?

Solving classic problems won't directly help *censorship victims* and *LastPass users*

Deep packet inspection systems can block protocols

- Ciphertexts don't "look like" benign traffic to network monitors

LastPass uses password-based encryption that can be cracked

- Decryption reveals when wrong key is used

Traditional approach: relegate such problems to systems security

Our approach: new symmetric encryption primitives

New symmetric encryption primitives can help *censorship victims* and *LastPass users*

Deep packet inspection systems can block protocols

FTE ciphertexts "look like" benign traffic to network monitors

LastPass uses password-based encryption that can be cracked

HE decryptions indistinguishable from real plaintexts

Traditional approach: relegate such problems to systems security

Our approach: new symmetric encryption primitives

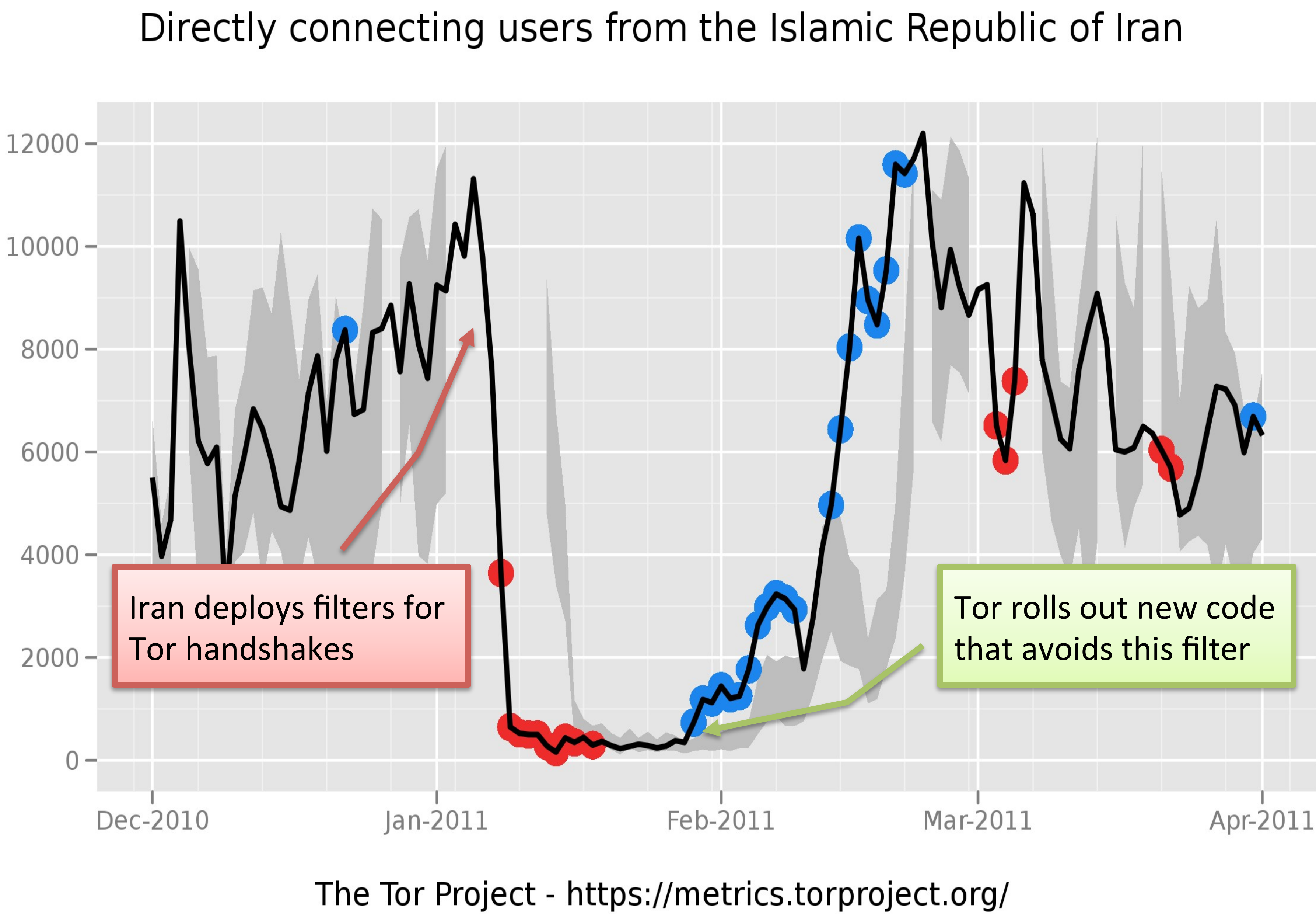
Today’s talk

- Part 1: Format-transforming encryption

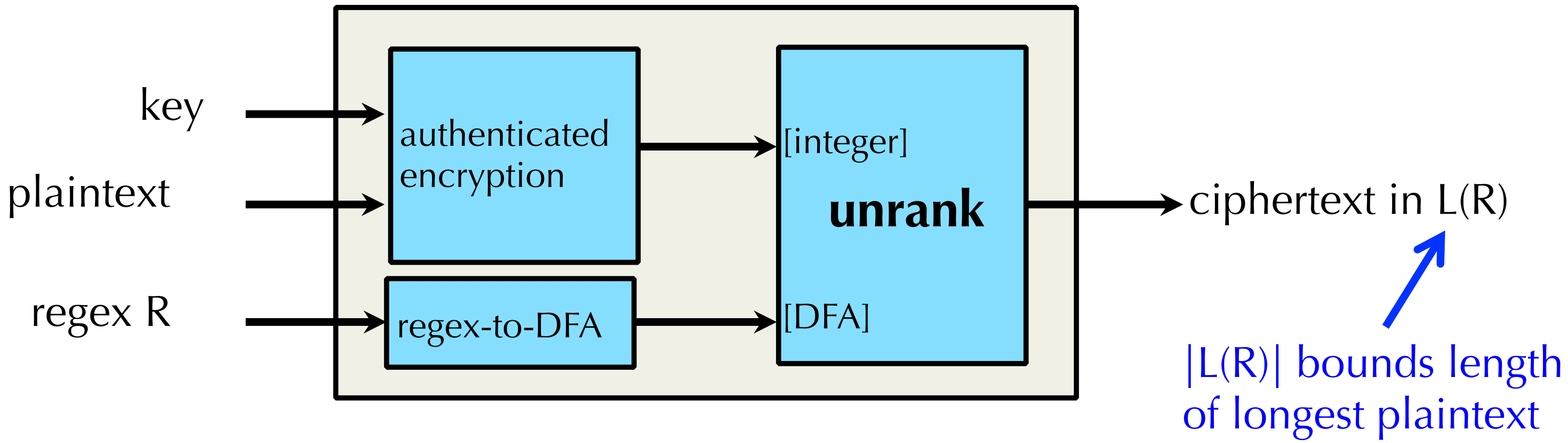
  [Dyer, Coull, R., Shrimpton – CCS 2013]

- Part 2: Honey encryption

  [Juels, R. – Eurocrypt 2014]


Directly connecting users from the Islamic Republic of Iran
12000
10000
8000
6000
4000
2000
0
Dec-2010
Jan-2011
Feb-2011
Mar-2011
Apr-2011
Iran deploys filters for Tor handshakes
Tor rolls out new code that avoids this filter
The Tor Project - https://metrics.torproject.org/

FTE engineering challenge: large plaintexts



Using very large languages leads to:

large tables – naively, (#DFA states) x (length of longest plaintext)

latency issues – waiting for long plaintext to buffer

Chunking, and using unrank(C_1), unrank(C_2), unrank(C_3), leads to:

receiver-side parsing issues – how to affect the commas?

Today's DPI evaded by FTE

Can DPI adapt to detect FTE?

Approach	Issues
Use R_1 , R_2 against FTE	False positives
Find R that matches against FTE, but not legitimate	Fast to change FTE formats
Find non-regular checks (e.g., HTTP Content-length field)	Speed? (~30% of Alexa traffic doesn't include Content-length)
???	???

Today's DPI evaded by FTE

Can DPI adapt to detect FTE?

Time	Protocol	Length	Info
0.010808	HTTP	301	GET /main/jobs/jobs/c3GcfOpJmL.png HTTP/1.0
0.888479	HTTP	15177	HTTP/1.1 200 OK (PNG)

```
▼ Hypertext Transfer Protocol
  ▶ GET /main/jobs/jobs/c3GcfOpJmL.png HTTP/1.0\r\n
    Host: htmlmusicsports.org\r\n
    Connection: close\r\n
    Date: Tue, 24 Aug 9446 97:07:16 PST\r\n
    If-Modified-Since: Wed, 19 Aug 1427 28:23:80 GMT\r\n
    Cookie: PHPSESSID=caC343B8b217bE8a759d21D3FDDcD5aa;\r\n
```