How cryptography relates to Internet censorship circumvention

David Fifield <david@bamsoftware.com>

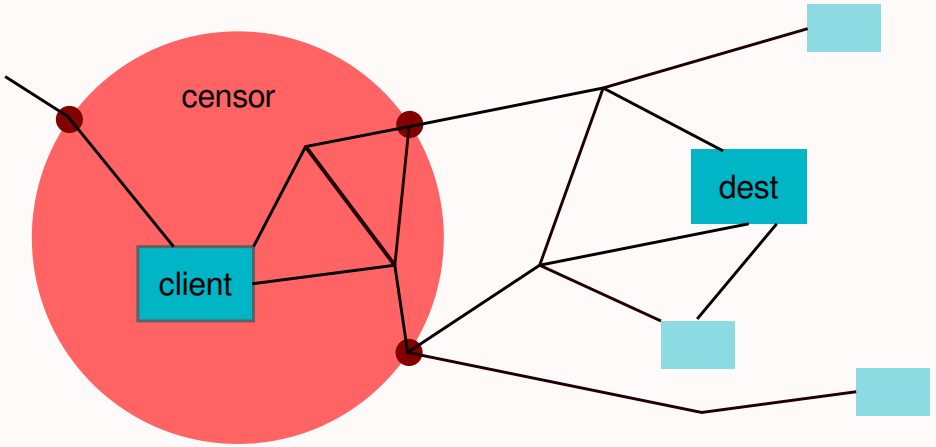
Workshop on Attacks in Cryptography 7
2024-08-18

https://www.bamsoftware.com/talks/wac7-fep/

1. Show some neat, but little-known, crypto-related attacks on significant protocols.
2. Talk about Fully Encrypted Protocols (FEPs).
3. Introduce censorship threat models to the cryptography-minded.

"Comments on certain past cryptographic flaws affecting fully encrypted censorship circumvention protocols" (2023)
David Fifield
https://eprint.iacr.org/2023/1362

Censorship and circumvention



A simple but fairly general model of Internet censorship.

The censor derives utility from permitting *some* forms of network access. ("Block everything" is not a profitable strategy.)

Circumvention tools work by making what the censor wants to allow, and what the censor wants to block, difficult to tell apart—by obfuscating the censor's decision boundary such that underblocking is more appealing than overblocking.

A censor may be understood as a combination of *detection* and *blocking* components.

Detection	Blocking
• Destination IP addresses	• IP null routing
• TLS SNI (Server Name Indication)	• TCP RST injection
• Contents of DNS queries	• DNS response injection
• Signatures of proxy protocols	• HTTP redirect injection
• Plaintext keywords	• Packet dropping
• …	• …

Generally, circumvention tools use some kind of *encrypted proxy*: encryption to hide the *content* of communication, and a proxy to hide its *endpoint*.

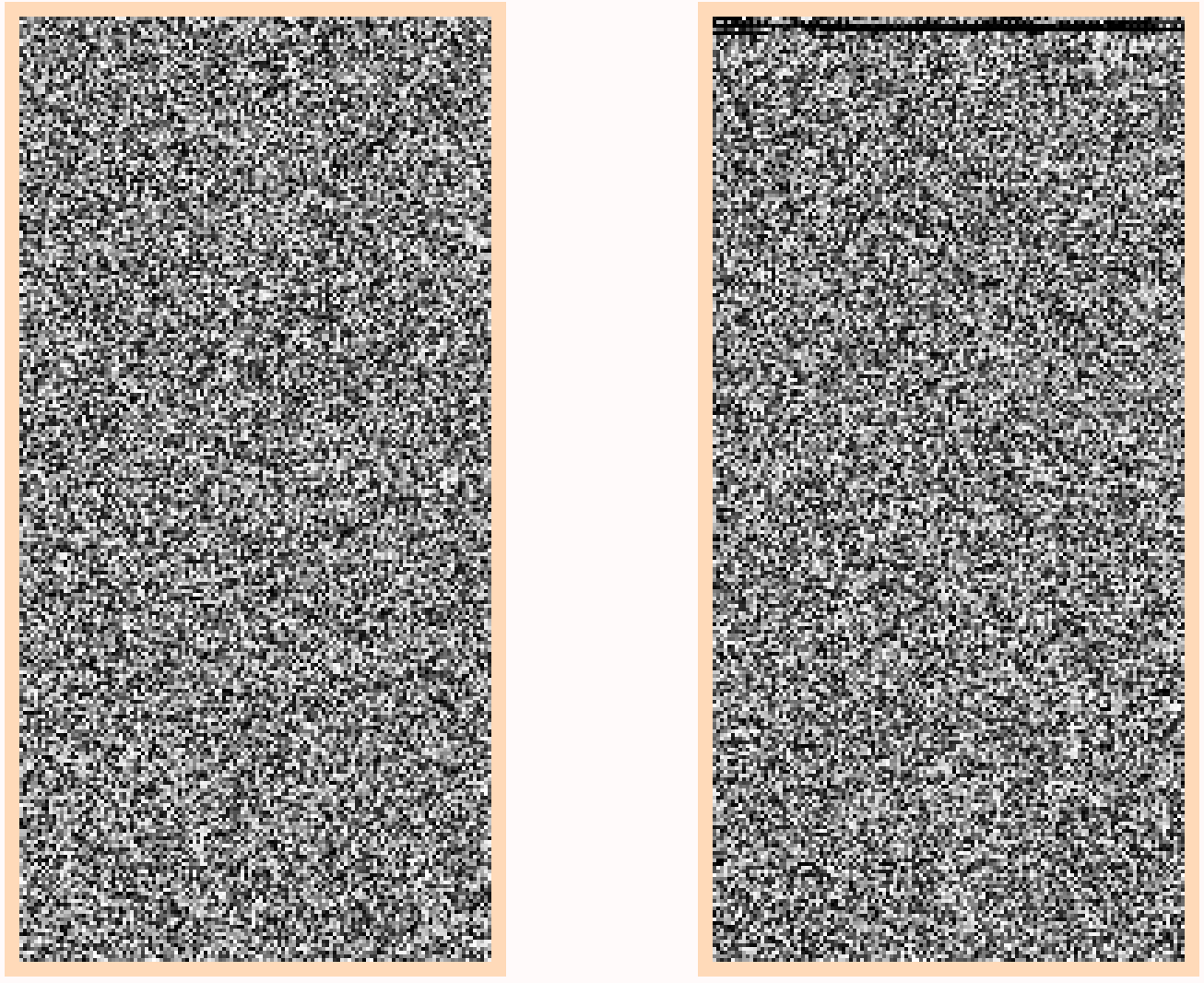
Fully Encrypted Protocols (FEPs)

Every byte in a connection appears independently and uniformly random.

"Security notions for fully encrypted protocols" (2023)
Ellis Fenske, Aaron Johnson
https://www.petsymposium.org/foci/2023/foci-2023-0004.php
https://github.com/net4people/bbs/issues/383

"Bytes to Schlep? Use a FEP: Hiding Protocol Metadata with Fully Encrypted Protocols" (2024)
Ellis Fenske, Aaron Johnson
https://arxiv.org/abs/2405.13310

"Obfuscated Key Exchange" (2024)
Felix Günther, Douglas Stebila, Shannon Veitch
https://eprint.iacr.org/2024/1086



Protocol bytes mapped to grayscale pixels.

FEP-like protocols have a long history in circumvention

Shadowsocks "stream ciphers" → Shadowsocks "AEAD ciphers"

Obfuscated OpenSSH (2009) → obfs2 (2012) → obfs3 (2013) → ScrambleSuit (2014) → obfs4 (2014)

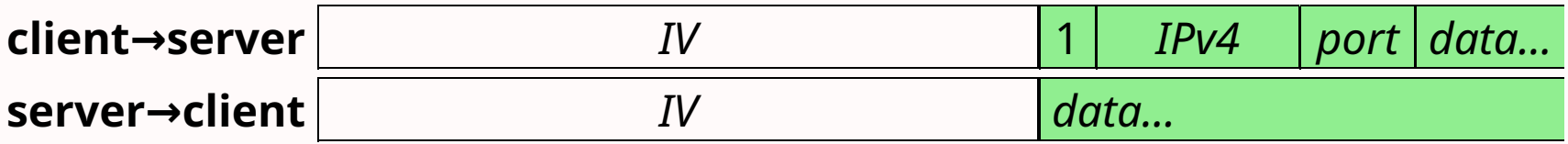
VMess

Shadowsocks stream ciphers decryption oracle

"Redirect attack on Shadowsocks stream ciphers"
Zhiniang Peng
https://github.com/edwardz246003/shadowsocks
https://github.com/net4people/bbs/issues/24

> ***Stream ciphers are completely broken and will be removed soon.***
> ***New users must use AEAD ciphers.***
> This historic document is for educational purposes only.
>
> https://shadowsocks.org/doc/stream.html

Client and server have a preshared symmetric key. The same key is used for encryption in both directions and across connections. Only the IV changes in different connections.



The server receives and starts decrypting the client's stream, checks for the magic byte 1, tries to make a TCP connection to the specified target at *IPv4*:*port*, then forwards the decryption of the rest of the stream.

The server encrypts any downstream data received from the server and sends it back to the client.

Idea: take a recorded server→client stream and replay it as a client→server stream.

The Shadowsocks server will interpret the first 7 bytes as a (*addrtype*, *IPv4*, *port*) target specification, decrypt the rest of the stream, and send it to that target.

This gives an attacker awkward *oracle access* to the server's secret key.

An attacker who can guess the first 7 bytes of plaintext (e.g. "HTTP/1.") can XOR its own IP address into the target specification and have the plaintext sent to a destination of its choosing.

Replay a recorded server→client stream to the server. (IV omitted here.)

```
7c20f534e986dbce37f555c6760ea24faa928f76
```

The server logs:

```
unsupported addrtype 72, maybe wrong password or encryption
method
```

```
  7c20f534e986dbce37f555c6760ea24faa928f76
⊕485454502f312e ("HTTP/1.")
⊕01cb0071051f40 (203.0.113.5:8000)
──────────────────────────────────────────
  35bfa115c3a8b5ce37f555c6760ea24faa928f76
```

"Why care about chosen ciphertext attacks? Won't it just decrypt to gibberish?"

"How would an attacker get access to a decryption oracle anyway?"

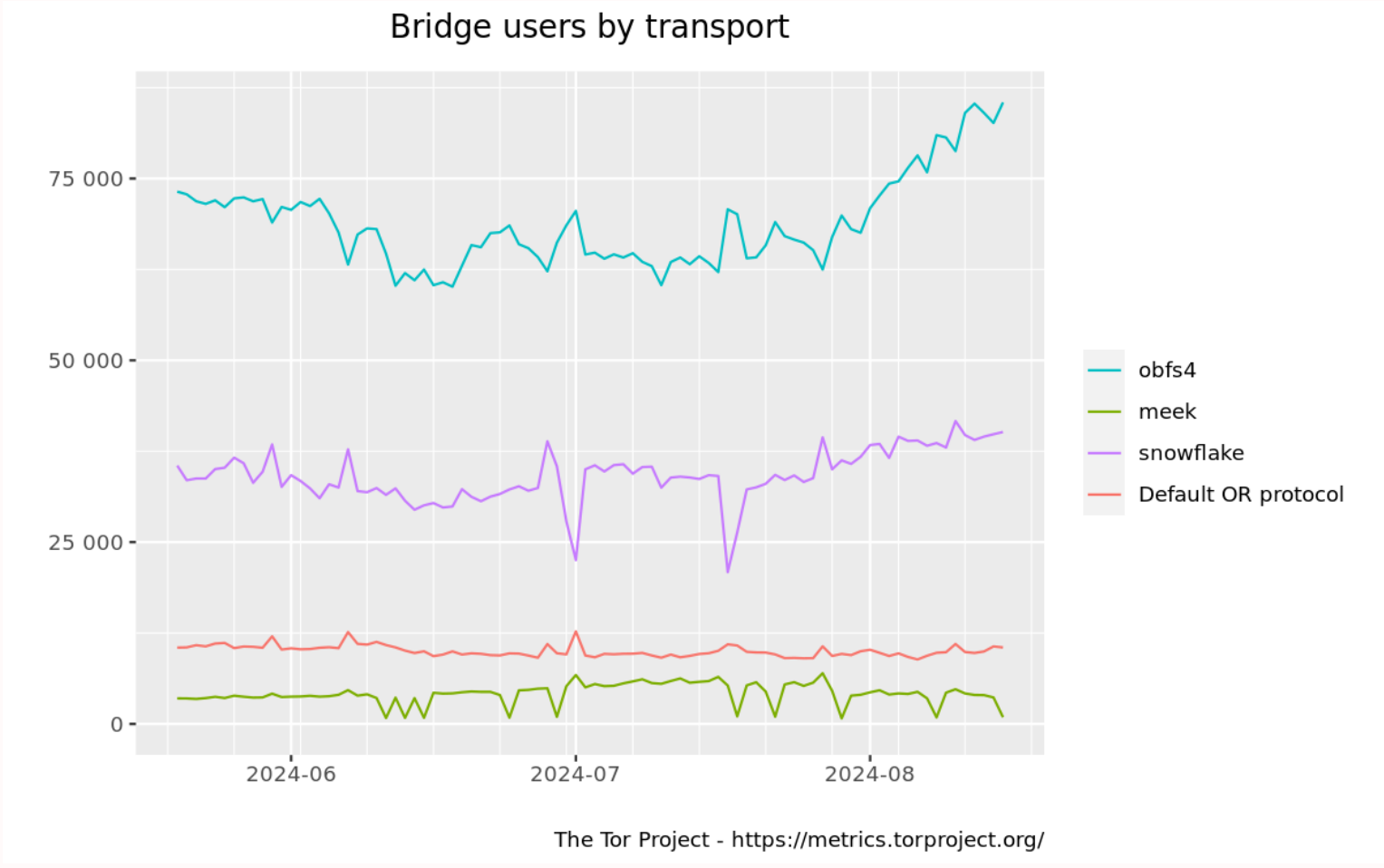
obfs4 and Elligator

obfs4

- Forward secrecy (X25519 handshake)
- Server authentication
- Integrity protection
- Probe resistance

Elligator

Represents public curve points (ephemeral public keys for Diffie–Hellman) as uniform random byte string "representatives".



Circumvention protocols used to access Tor. (Tor Metrics.)

obfs4 begins with an elliptic curve Diffie–Hellman key exchange. Client and server exchange ephemeral public keys from which a shared session key is derived.

Public keys are sent as Elligator representatives, so they look as random as the rest of the protocol.

1. Non-canonical public key representatives
2. Most significant bit always zero
3. Only points from the large prime-order subgroup

Non-canonical representatives

The final step of the Elligator inverse map is to take a square root in GF(q), where $q = 2^{255} - 19$.

The square root is supposed to produce only non-negative outputs (e.g., field elements in $[0, (q - 1)/2]$).

The implementation of Elligator used at the time did not use canonical square roots, and instead systematically produced either a positive or negative square root for a given input. (In other words, bit 254 was not independent of the lower bits.)

"Implementing Elligator for Curve25519" (2013)
Adam Langley
https://www.imperialviolet.org/2013/12/25/elligator.html

The attack: observe a possible public key representative at the beginning of a suspected obfs4 connection.

Interpret the bytes as a field element, square it, and take the square root using the same non-canonical square root algorithm.

The output will match the input only 50% of the time for random byte strings, but 100% of the time for the non-canonical representatives.

"Incorrect (non canonical) representative output for ScalarBaseMult()" (2020)
Loup Vaillant
https://github.com/agl/ed25519/issues/27

Most significant bit always zero

Elligator outputs 254-bit representatives (when the canonical square root is used). But programming interfaces are more naturally expressed using *bytes*, not bits.

Unclear API boundaries meant that nothing was randomizing the remaining 2 bits of the `byte[32]`.

```
$ perl -an -e '/^obfs4 (\S*) (?:\S+) cert=(\S+)/ && \
print "$1 $2\n";' bridges.txt | while read addr cert; do \
./obfs4-bug-check $addr $cert; done
..................... 185.31.174.60:443 FAIL 0/20
..................... 142.132.237.143:2538 FAIL 0/20
1.111...1...1..11..1 90.127.32.238:42024 PASS 9/20
```

https://bugs.torproject.org/tpo/anti-censorship/team/91

Only points from the large prime-order subgroup

Curve25519 has order 8*p* for a large prime *p*. To avoid small-subgroup attacks, the base point of X25519 generates the subgroup of order *p*. Also, private key scalars are "clamped" to be multiples of 8.

But Elligator representatives must *not* always represent points on the order-*p* subgroup, because random byte strings do not have that property.

The passive attack is to observe a suspected obfs4 handshake, map the public key representative to an elliptic curve point, multiply by *p*, and check for the result always being the identity element.

https://bugs.torproject.org/tpo/anti-censorship/pluggable-transports/lyrebird/40007

If you're doing any implementation with Elligator, your first stop should be https://elligator.org/.

The effect of these attacks

Shadowsocks stream ciphers are dead (one hopes), but Shadowsocks is otherwise still going strong.

obfs4 implementations were patched and obfs4 continues to be a workhorse in Tor and elsewhere. There's no evidence of any of these passive obfs4 distinguishability attacks having been used in practice.

Fully encrypted protocols remain probably the most important single class of circumvention protocols.

"How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic" (2023)
Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, Eric Wustrow
https://gfw.report/publications/usenixsecurity23/en/

Why?

In circumvention, the crypto is usually not the weak link. Available evidence indicates that censors prefer to attack circumvention systems in other ways, when possible.

Censors are not just classifiers—they also bear asymmetric *costs* for misclassifications.

Classification advantage is asymmetric: censors are highly sensitive to false positives. A classifier FPR of 0.1% spells a successful circumvention protocol.

Corollary: censors almost universally prefer allow-by-default rather than deny-by-default, fail open rather than fail closed.

Censors, empirically, dislike employing probabilistic or stateful attacks like the obfs4 passive distinguishers.

Shadowsocks, for its shortcomings, is easy to specify and easy to implement, and for those reasons has huge support and momentum.

> …worse-is-better has better survival characteristics than the-right-thing…
>
> https://dreamsongs.com/RiseOfWorseIsBetter.html

Circumvention technology stands to benefit from better cryptography, but we must still have due respect for the other factors that matter.

jīchǎng

机场

"Airports" are paid Shadowsocks/etc. hosting services that are common and commonly used in China.

Shadowsocks logo.

Why doesn't a simple TLS connection to a proxy suffice?

- TLS fingerprinting
- Active probing
- But the big one is: how do you prevent the censor from discovering the IP addresses of your proxies and simply blocking them by IP address?

In practice, the biggest challenge is not protocol obfuscation but *endpoint blocking*: the "bridge distribution problem".

"Lox: Protecting the Social Graph in Bridge Distribution" (2023)
Lindsey Tulloch, Ian Goldberg
https://censorbib.nymity.ch/#Tulloch2023a
https://github.com/net4people/bbs/issues/320

David Fifield <david@bamsoftware.com>

The censorship and circumvention forum I manage: https://github.com/net4people/bbs