SSL logging for proof

Why?
Money hardness scale:
BTC 10 (+!)
Physical cash - 10
Wire transfer over SWIFT – 10 (?)
OK Pay – 10(?)
SEPA, ACH – 5-8?
Paypal, credit cards – 0

So – swap BTC for wire transfers or OK Pay, or maybe “softer” methods like SEPA if small amounts.

Localbitcoins – peer to peer but drawbacks:
Safety/counterfeiting and physical inconvenience if in person for cash
Quasi-centralisation and lack of effective proof if online for wire transfer
(Quality of dispute arbitration is dubious)

Other “Whys” - there can be a few other ways to use this technology.
These slides only talk about trading BTC.

SSL logging for proof

Record ssl session between bank and customer (*NOT* MITM)
If there's a dispute, the customer can prove the payment was made
by providing decryption key(s) to an arbitrator.

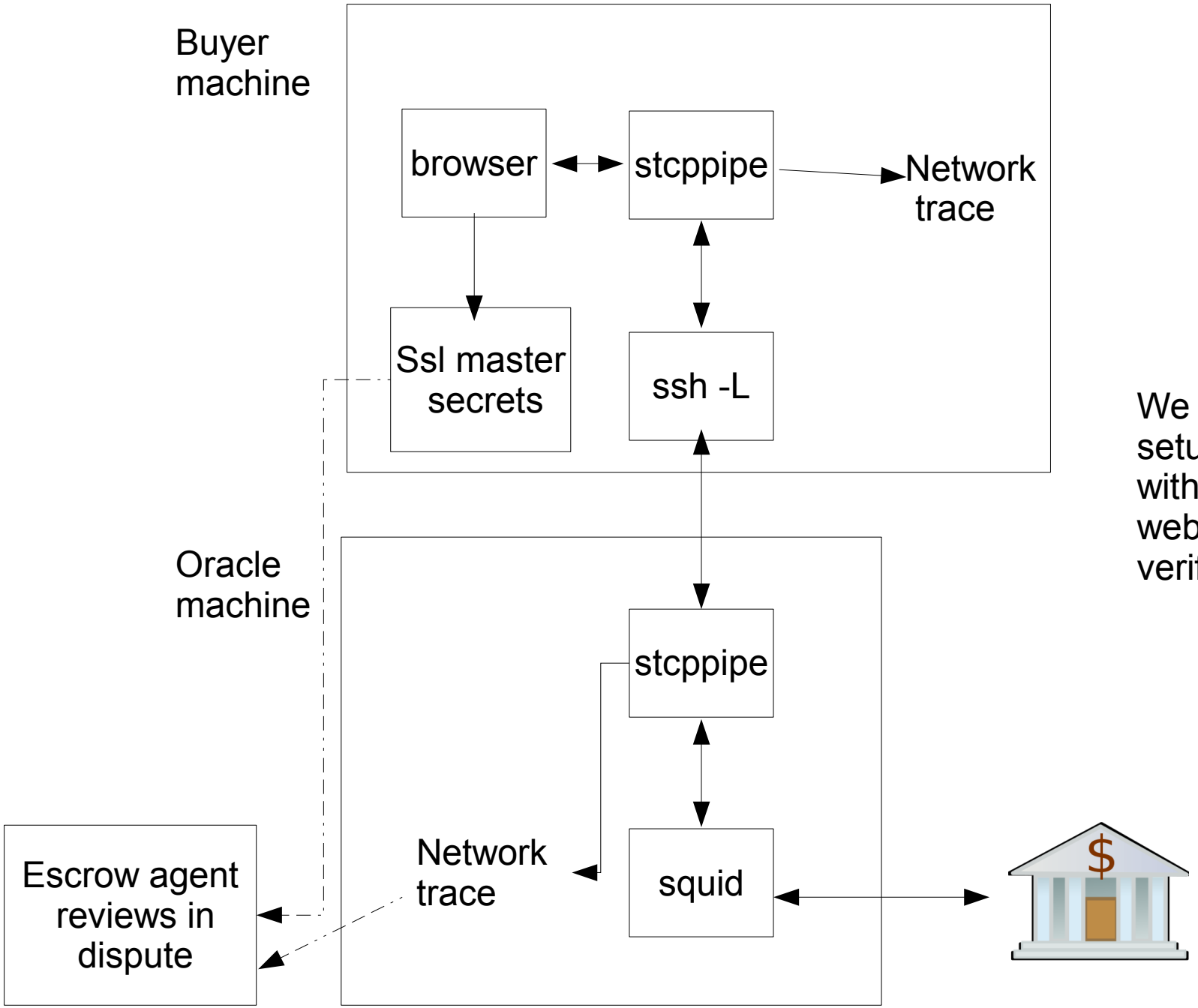
Privacy issues:
It's possible to break the SSL connection to the bank and then rebuild it. Thus, the customer only provides the decryption for 1 or 2 HTML pages containing proof of payment, not the whole banking session. So login information need not be exposed.

Network trace

Wireshark – open source, extremely rich, long running industry standard (originally “Ethereal”).
Command line version “tshark”.
Recently added facility to decrypt ssl sessions if given master secret
Mozilla NSS library recently added facility to record master secret via an env variable SSLKEYLOGFILE – works with Firefox and maybe Chrome.
Network traces recorded with Wireshark are flaky w.r.t. ssl decryption (issues with e.g. out of order packets).
Much better results using traces recorded with stcppipe by Luigi Ariemma - “simple tcp pipe”.

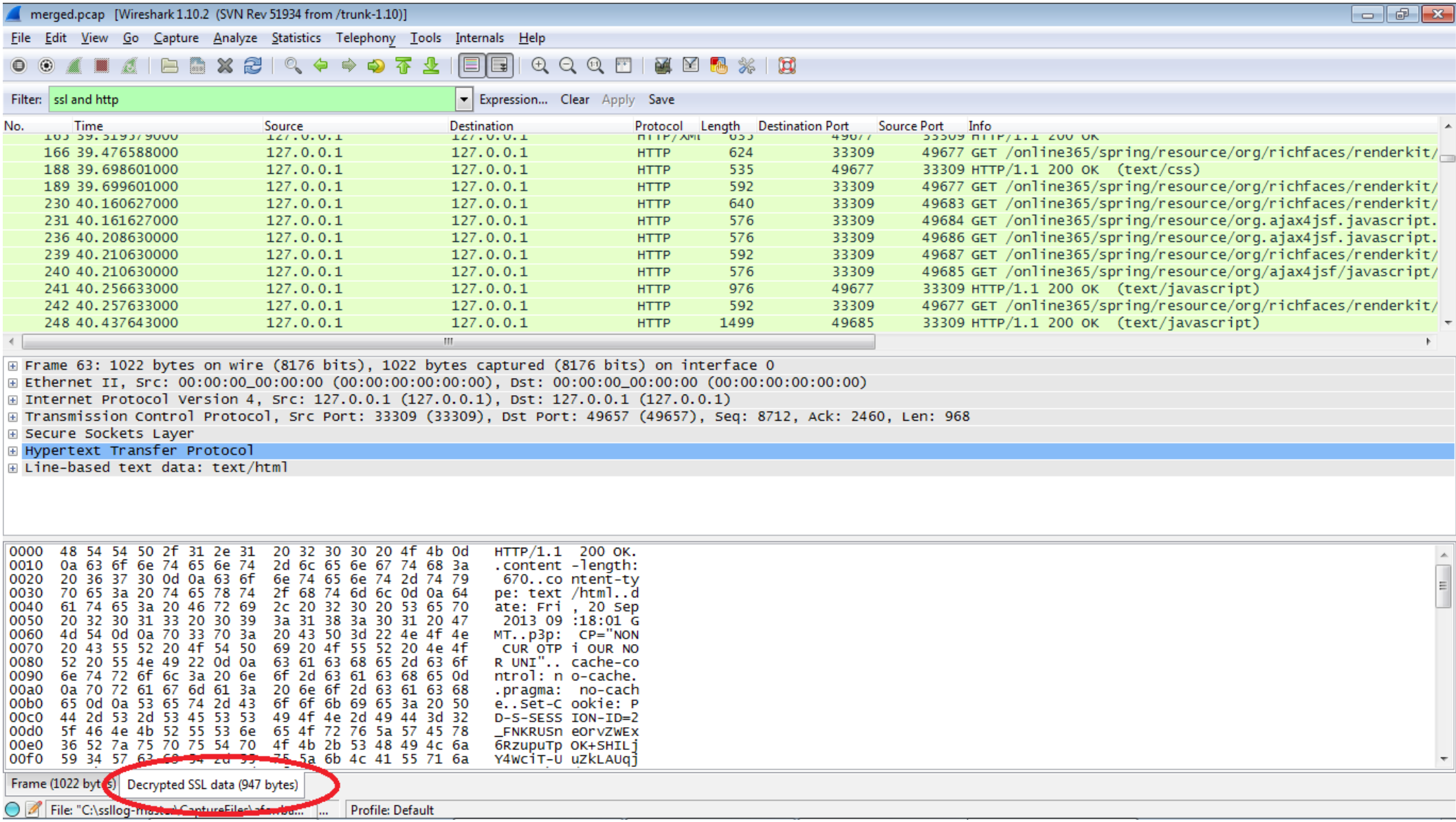
All of this cross platform and open source.

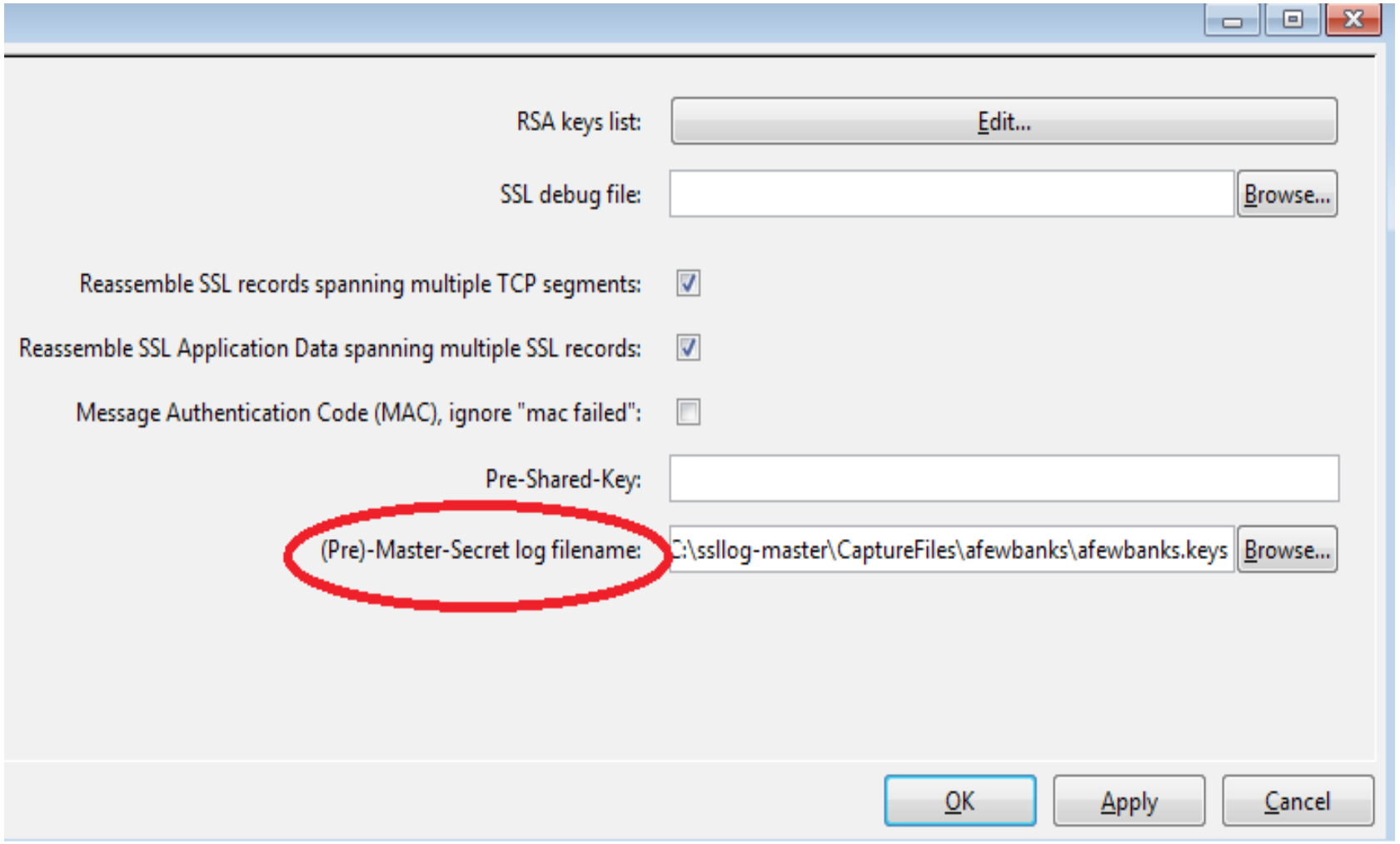
Side note: Wireshark doesn't seem to be able to decrypt Diffie Helman ciphers, although it should be able to in theory (perfect forward secrecy shouldn't affect this).
Camellia ciphers also appear not to work. These can be switched off in FF prefs.



We have tested this setup
with several bank websites and
verified it works.

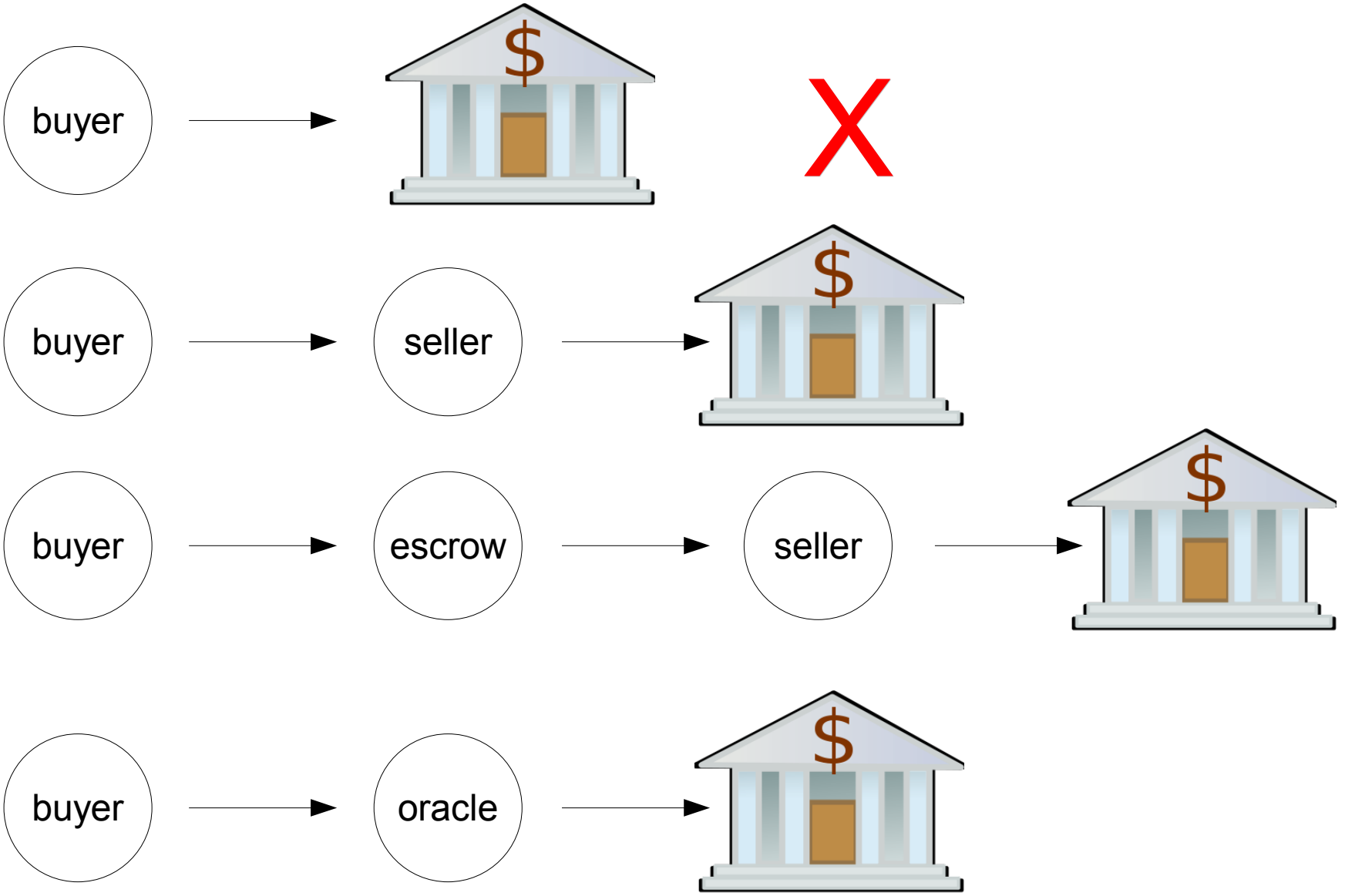
Wireshark - decryption






```
Example usage:
>>tshark -r /path/to/tracefile -Y \
“ssl and http.content_type contains html” \
-o ssl.keylog_file:/path/to/keylogfile \
-T fields -e frame.number
```

Architectures



Step by step

- Whichever exact model is chosen, the basic structure would be:
- Buyer and seller negotiate a TX
- 2 of 3 multisig address is created and funded by seller - 3rd key belongs to escrow agent
- Seller funds multisig
- Buyer does banking session
- If no dispute, buyer and seller send TX to network
- If dispute, escrow agent requests ssl key for the 1 or 2 HTML pages constituting proof of payment
- Escrow can make a valid decision and sign a transaction either to send to buyer or seller

Issues

- NAT traversal for buyer-seller direct connection; would need proper P2P network
- In order to arbitrate, the escrow needs to know who is lying if the network trace can't be decrypted; that rules out simple seller-as-proxy model
- Should we run this complex auditing process **during** the bank payment or **after** – only in case of dispute? Not a trivial question!
- Oracle implemented in the cloud so that it's known what code is running on it – so it can be trusted. BUT – we can't have tons of bank session traffic coming just from Amazon AWS IPs (geographical and other restrictions)

New ideas

- Master secret is partitioned into 4 16 byte blocks – one block each for encryption key and mac key (for message authentication), one pair for client, one for server (TLS RFC 6.3)
- This means we can have the buyer able to decrypt/encrypt but not able to authenticate – we just need to withhold the server mac key and disable the mac check.
- This would provide another party the ability to verify the validity of the buyer's traffic after the event.
- This is a very powerful idea, although it does have security implications