BTI 4202: Security and Trust in Distributed Systems

Christian Grothoff

Berner Fachhochschule

13.5.2022
Risk Analysis: Operating a Tor Hidden Service
Learning objectives

Fallacies of distributed computing

Boyd’s theorem

CAP Theorem

Zooko’s Triangle

Self stabilization

Attacks and defenses

Distributed Hash Tables
  CAN
  Chord
  Kademlia

Advanced Cryptographic Primitives

Secure Multiparty Computation
Part I: Security in Distributed Systems
The 8 Fallacies of Distributed Computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology does not change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

According to Peter Deutsch and James Gosling
Limits on authentication

Theorem (Boyd’s Theorem I)

“Suppose that a user has either a confidentiality channel to her, or an authentication channel from her, at some state of the system. Then in the previous state of the system such a channel must also exist. By an inductive argument, such a channel exists at all previous states.”

Theorem (Boyd’s Theorem II)

“Secure communication between any two users may be established by a sequence of secure key transfers if there is a trusted chain from each one to the other.”
Solution space: Zfone Authentication (ZRTP) [5]

Idea: combine human interaction proof and baby duck approach:

- $A$ and $B$ perform Diffie-Hellman exchange
- Keying material from previous sessions is used (duckling)
- Short Authentication String (SAS) is generated (hash of DH numbers)
- Both users read the SAS to each other, recognize voice

⇒ ZRTP foils standard man-in-the-middle attack.
CAP Theorem [3]

No distributed system can be consistent, available and partition tolerant at the same time.

- **Consistency**: A read sees the changes made by all previous writes
- **Availability**: Reads and writes always succeed
- **Partition tolerance**: The system operates even when network connectivity between components is broken
Blockchain Trilemma

Blockchains claim to achieve three properties:

- **Decentralization:** there are many participants, and each participant only needs to have a small amount of resources, say $O(c)$
- **Scalability:** the system scales to $O(n) > O(c)$ transactions
- **Security:** the system is secure against attackers with $O(n)$ resources

The Blockchain trilemma is that one can only have two of the three.
Ryge’s Triangle postulates three key management goals for a system associating cryptographic keys with addresses or names:

- Non-interactive: the system should require no user interface
- Flexible: addresses/names can be re-used by other participants
- Secure: the system is secure against active attackers

Ryge’s triangle says that one can only have two of the three.
A name system can only fulfill two!
Zooko’s Triangle

- Secure
- Global
- Hierarchical Registration
- Cryptographic Identifiers
- Memorable
- Petname Systems

DNS, “.onion” IDs and /etc/hosts/ are representative designs.
Zooko’s Triangle

DNSSEC security is limited (adversary model!)
Self stabilization (Dijkstra 1974)

- A system is self-stabilizing, if starting from any state, it is guaranteed that the system will eventually reach a correct state (convergence).
- Given that the system is in a correct state, it is guaranteed to stay in a correct state, provided that no fault happens (closure).
- Self-stabilization enables a distributed algorithm to recover from a transient fault regardless of its nature.

Example: Spanning-tree Protocol from Networking!
Sybil Attack

Background:
- Ancient Greece: Sybils were prophetesses that prophesized under the divine influence of a deity. Note: At the time of prophecy not the person but a god was speaking through the lips of the sybil.
Sybil Attack

Background:

▶ Ancient Greece: Sybils were prophetesses that prophesized under the divine influence of a deity. Note: At the time of prophecy not the person but a god was speaking through the lips of the sybil.


The Sybil Attack [2]:

▶ Insert a node multiple times into a network, each time with a different identity

▶ Position a node for next step on attack:
  ▶ Attack connectivity of the network
  ▶ Attack replica set
  ▶ In case of majority votes, be the majority.
Defenses against Sybil Attacks

- Use authentication with trusted party that limits identity creation
- Use “external” identities (IP address, MAC, e-mail)
- Use “expensive” identities (solve computational puzzles, require payment)

Douceur: Without trusted authority to certify identities, no realistic approach exists to completely stop the Sybil attack.
Eclipse Attack: Goal

- Separate a node or group of nodes from the rest of the network
- Isolate peers (DoS, surveillance) or isolate data (censorship)
Eclipse Attack: Techniques

- Use Sybil attack to increase number of malicious nodes
- Take over routing tables, peer discovery
  ⇒ Details depend on overlay structure
Eclipse Attack: Defenses

- Large number of connections
- Replication
- Diverse neighbour selection (different IP subnets, geographic locations)
- Aggressive discovery ("continuous" bootstrap)
- Audit neighbour behaviour (if possible)
- Prefer long-lived connections / old peers
Poisoning Attacks

Nodes provide false information:

- wrong routing tables
- wrong meta data
- wrong performance measurements
Nodes can:
- measure latency to determine origin of data
- delay messages
- send messages using particular timing patterns to aid correlation
- include wrong timestamps (or just have the wrong time set...)

Timing Attacks [4]
Break
Part II: Distributed Hash Tables
Distributed Hash Tables (DHTs)

- Distributed index
- GET and PUT operations like a hash table
- JOIN and LEAVE operations (internal)
- Trade-off between JOIN/LEAVE and GET/PUT costs
- Typically use exact match on cryptographic hash for lookup
- Typically require overlay to establish particular connections
DHTs: Key Properties

To know a DHT, you must know (at least) its:

- routing table structure
- lookup procedure
- join operation process
- leave operation process

... including expected costs (complexity) for each of these operations.
A trivial DHTs: The Clique

- routing table: hash map of all peers
- lookup: forward to closest peer in routing table
- join: ask initial contact for routing table, copy table, introduce us to all other peers, migrate data we’re closest to to us
- leave: send local data to remaining closest peer, disconnect from all peers to remove us from their routing tables

Complexity?
A trivial DHTs: The Circle

- routing table: left and right neighbour in cyclic identifier space
- lookup: forward to closest peer (left or right)
- join: lookup own peer identity to find join position, transfer data from neighbour for keys we are closer to
- leave: ask left and right neighbor connect directly, transfer data to respective neighbour

Complexity?
Additional Questions to ask

- Security against Eclipse attack?
- Survivability of DoS attack?
- Maintenance operation cost & required frequency?
- Latency? \(\neq\) number of hops!
- Data persistence?
Content Addressable Network: CAN

▶ routing table: neighbours in \(d\)-dimensional torus space
▶ lookup: forward to closest peer
▶ join: lookup own peer identity to find join position, split quadrant (data areas) with existing peer
▶ leave: assign quadrant space to neighbour (s)
Interesting CAN properties

- CAN can do range queries along $\leq n$ dimensions
- CAN's peers have $2d$ connections (independent of network size)
- CAN routes in $O(d^{\sqrt{n}})$
Chord

- routing table: predecessor in circle and at distance $2^i$, plus $r$ successors
- lookup: forward to closest peer (peer ID after key ID)
- join: lookup own peer identity to find join position, use neighbor to establish finger table, migrate data from respective neighbour
- leave: join predecessor with successor, migrate data to respective neighbour, periodic stabilization protocol takes care of finger updates
Interesting Chord properties

- Simple design
- $\log_2 n$ routing table size
- $\log_2 n$ lookup cost
- Asymmetric, inflexible routing tables
Kademlia

- routing table: $2^{160}$ buckets with $k$ peers at XOR distance $2^i$
- lookup: iteratively forward to $\alpha$ peers from the “best” bucket, selected by latency
- join: lookup own peer identity, populate table with peers from iteration
- maintenance: when interacting with a peer, add to bucket if not full; if bucket full, check if longest-not-seen peer is live first
- leave: just drop out

```
Connections
Route path
```

```
0 1
0 1
10
 11
0 1
00
 01
```
Interesting Kademlia properties

- XOR is a symmetric metric: connections are used in both directions
- $\alpha$ replication helps with malicious peers and churn
- Iterative lookup gives initiator much control,
- Lookup helps with routing table maintenance
- Bucket size trade-off between routing speed and table size
- Iterative lookup is a trade-off:
  - good UDP (no connect cost, initiator in control)
  - bad with TCP (very large number of connections)
Break
Homomorphic Encryption

\[ E(x_1 \oplus x_2) = E(x_1) \otimes E(x_2) \] (1)
Multiplicative Homomorphism: RSA & ElGamal

- Unpadded RSA (multiplicative):
  \[ E(x_1) \cdot E(x_2) = x_1^e x_2^e = E(x_1 \cdot x_2) \]  \hspace{1cm} (2)

- ElGamal:
  \[ E(x_1) \cdot E(x_2) = (g^{r_1}, x_1 \cdot h^{r_1})(g^{r_2}, x_2 \cdot h^{r_2}) \]  \hspace{1cm} (3)
  \[ = (g^{r_1+r_2}, (x_1 \cdot x_2)h^{r_1+r_2}) \]  \hspace{1cm} (4)
  \[ = E(m_1 \cdot m_2) \]  \hspace{1cm} (5)
Additive Homomorphism: Paillier

\[ E_K(m) : = g^m \cdot r^n \mod n^2, \quad (6) \]
\[ D_K(c) : = \frac{(c^\lambda \mod n^2) - 1}{n} \cdot \mu \mod n \quad (7) \]

where the public key \( K = (n, g) \), \( m \) is the plaintext, \( c \) the ciphertext, \( n \) the product of \( p, q \in \mathbb{P} \) of equal length, and \( g \in \mathbb{Z}_n^* \). In Paillier, the private key is \( (\lambda, \mu) \), which is computed from \( p \) and \( q \) as follows:

\[ \lambda : = \text{lcm}(p - 1, q - 1), \quad (8) \]
\[ \mu : = \left( \frac{(g^\lambda \mod n^2) - 1}{n} \right)^{-1} \mod n. \quad (9) \]

Paillier offers additive homomorphic public-key encryption, that is:

\[ E_K(a) \otimes E_K(b) \equiv E_K(a + b) \quad (10) \]

for any public key \( K \).
Fully homomorphic encryption

Additive:

\[ E(A) \oplus E(B) = E(A + B) \]  \hspace{1cm} (11)

and multiplicative:

\[ E(A) \otimes E(B) = E(A \cdot B) \]  \hspace{1cm} (12)

Known cryptosystems: Brakerski-Gentry-Vaikuntanathan (BGV), NTRU, Gentry-Sahai-Waters (GSW).
Let $G_1$, $G_2$ be two additive cyclic groups of prime order $q$, and $G_T$ another cyclic group of order $q$ (written multiplicatively). A pairing is an efficiently computable map $e$:

$$e : G_1 \times G_2 \rightarrow G_T$$

which satisfies $e \neq 1$ and bilinearity:

$$\forall a, b \in \mathbb{F}_q^*, \forall P \in G_1, Q \in G_2 : e(aP, bQ) = e(P, Q)^{ab}$$

Examples: Weil pairing, Tate pairing.
Hardness assumption

Computational Diffie Hellman:

\[ g, g^x, g^y \Rightarrow g^{xy} \] (15)

remains hard on \( G \) even given \( e \).
Boneh-Lynn-Sacham (BLS) signatures [1]

Key generation:
   Pick random \( x \in \mathbb{Z}_q \)

Signing:
   \( \sigma := h^x \) where \( h := H(m) \)

Verification:
   Given public key \( g^x \):
   \[
   e(\sigma, g) = e(h, g^x)
   \] (16)
Boneh-Lynn-Sacham (BLS) signatures [1]

Key generation:
Pick random $x \in \mathbb{Z}_q$

Signing:
$$\sigma := h^x \text{ where } h := H(m)$$

Verification:
Given public key $g^x$:
$$e(\sigma, g) = e(h, g^x)$$  \hspace{1cm} (16)

Why:
$$e(\sigma, g) = e(h, g^x) = e(h, g^x)$$  \hspace{1cm} (17)
due to bilinearity.
Given signature \( \langle \sigma, g^x \rangle \) on message \( h \), we can *blind* the signature and public key \( g^x \):

\[
e(\sigma^b, g) = e(h, g)^{xb} = e(h, g^{xb})
\]  

(18)

Thus \( \sigma^b \) is a valid signature for the *derived* public key \((g^x)^b\) with blinding value \( b \in \mathbb{Z}_q \).
Part III: Secure Multiparty Computation
Alice und Bob haben private Daten $a_i$ and $b_i$.

Alice und Bob führen ein Protokoll aus und berechnen gemeinsam $f(a_i, b_i)$.

Nur einer von beiden lernt das Ergebnis (i.d.R.)
Adversary models

Honest but curious

Dishonest and curious
Secure Multiparty Computation: Scalar Product

- Original idea by Ioannids et al. in 2002 [?] (use: \((a - b)^2 = a^2 - 2ab + b^2\))
- Refined by Amirbekyan et al. in 2007 (corrected math) [?]
- Now providing protocol with practical extensions (negative numbers, small numbers, set intersection).
Alice has public key $A$ and input map $m_A : M_A \to \mathbb{Z}$.
Bob has public key $B$ and input map $m_B : M_B \to \mathbb{Z}$.
We want to calculate
\[
\sum_{i \in M_A \cap M_B} m_A(i)m_B(i)
\] (19)
We first calculate $M = M_A \cap M_B$.
Define $a_i := m_A(i)$ and $b_i := m_B(i)$ for $i \in M$.
Let $s$ denote a shared static offset.
Alice transmits $E_A(s + a_i)$ for $i \in M$ to Bob.

Bob creates two random permutations $\pi$ and $\pi'$ over the elements in $M$, and a random vector $r_i$ for $i \in M$ and sends

$$R := E_A(s + a_{\pi(i)}) \otimes E_A(s - r_{\pi(i)} - b_{\pi(i)})$$

$$= E_A(2 \cdot s + a_{\pi(i)} - r_{\pi(i)} - b_{\pi(i)}),$$

(20)

$$R' := E_A(s + a_{\pi'(i)}) \otimes E_A(s - r_{\pi'(i)})$$

$$= E_A(2 \cdot s + a_{\pi'(i)} - r_{\pi'(i)}),$$

(22)

$$S := \sum (r_i + b_i)^2,$$

(24)

$$S' := \sum r_i^2$$

(25)
Decryption (1/3)

Alice decrypts $R$ and $R'$ and computes for $i \in M$:

$$a_{\pi(i)} - b_{\pi(i)} - r_{\pi(i)} = DA(R) - 2 \cdot s, \quad (26)$$
$$a_{\pi'(i)} - r_{\pi'(i)} = DA(R') - 2 \cdot s, \quad (27)$$

which is used to calculate

$$T : = \sum_{i \in M} a_i^2 \quad (28)$$
$$U : = - \sum_{i \in M} (a_{\pi(i)} - b_{\pi(i)} - r_{\pi(i)})^2 \quad (29)$$
$$U' : = - \sum_{i \in M} (a_{\pi'(i)} - r_{\pi'(i)})^2 \quad (30)$$
Decryption (2/3)

She then computes

\[ P : = S + T + U \]

\[ = \sum_{i \in M} (b_i + r_i)^2 + \sum_{i \in M} a_i^2 + \left( -\sum_{i \in M} (a_i - b_i - r_i)^2 \right) \]

\[ = \sum_{i \in M} ((b_i + r_i)^2 + a_i^2 - (a_i - b_i - r_i)^2) \]

\[ = 2 \cdot \sum_{i \in M} a_i (b_i + r_i). \]

\[ P' : = S' + T + U' \]

\[ = \sum_{i \in M} r_i^2 + \sum_{i \in M} a_i^2 + \left( -\sum_{i \in M} (a_i - r_i)^2 \right) \]

\[ = \sum_{i \in M} (r_i^2 + a_i^2 - (a_i - r_i)^2) = 2 \cdot \sum_{i \in M} a_i r_i. \]
Finally, Alice computes the scalar product using:

\[
\frac{P - P'}{2} = \sum_{i \in M} a_i (b_i + r_i) - \sum_{i \in M} a_i r_i = \sum_{i \in M} a_i b_i.
\]  

(31)
Who said calculating DLOG was hard?
Computing Discrete Logarithms
Giant Steps

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
</tr>
<tr>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
<td>106</td>
<td>107</td>
<td>108</td>
<td>109</td>
<td>110</td>
</tr>
<tr>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
<td>118</td>
<td>119</td>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
</tr>
<tr>
<td>128</td>
<td>129</td>
<td>130</td>
<td>131</td>
<td>132</td>
<td>133</td>
<td>134</td>
<td>135</td>
<td>136</td>
<td>137</td>
<td>138</td>
<td>139</td>
<td>140</td>
<td>141</td>
<td>142</td>
</tr>
</tbody>
</table>
Alice's geheimer Wert sei \( a \). Alice schickt an Bob \((g_i, h_i) = (g^{r_i}, g^{r_ia+a_i})\) mit zufälligen Werten \( r_i \) für \( i \in M \).

Bob antwortet mit

\[
\left( \prod_{i \in M} g_i^{b_i}, \prod_{i \in M} h_i^{b_i} \right) = \left( \prod_{i \in M} g_i^{b_i}, \left( \prod_{i \in M} g_i^{b_i} \right)^a g \sum_{i \in M} a_i b_i \right)
\]

Alice kann dann berechnen

\[
\left( \prod_{i \in M} g_i^{b_i} \right)^{-a} \cdot \left( \prod_{i \in M} g_i^{b_i} \right)^a \cdot g \sum_{i \in M} a_i b_i = g \sum_{i \in M} a_i b_i.
\]

Falls \( \sum_{i \in M} a_i b_i \) ausreichend klein ist, kann Alice dann das Skalarprodukt durch Lösung des DLP bestimmen.

\({}^2\) Joint work with Tanja Lange
The pre-calculation of ECC-$2^{28}$ is $\times 16$ more expensive than for ECC-$2^{20}$ as the table is set to have size $\sqrt{n}$. 

<table>
<thead>
<tr>
<th>Length</th>
<th>RSA-2048</th>
<th>ECC-$2^{20}$</th>
<th>ECC-$2^{28}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>4 s</td>
<td>0.1 s</td>
<td>4.2 s</td>
</tr>
<tr>
<td>50</td>
<td>8 s</td>
<td>0.1 s</td>
<td>4.3 s</td>
</tr>
<tr>
<td>100</td>
<td>10 s</td>
<td>0.2 s</td>
<td>4.3 s</td>
</tr>
<tr>
<td>200</td>
<td>19 s</td>
<td>0.2 s</td>
<td>4.3 s</td>
</tr>
<tr>
<td>400</td>
<td>35 s</td>
<td>0.3 s</td>
<td>4.3 s</td>
</tr>
<tr>
<td>800</td>
<td>74 s</td>
<td>0.4 s</td>
<td>4.5 s</td>
</tr>
<tr>
<td>800</td>
<td>1234 kb</td>
<td>65 kb</td>
<td>65 kb</td>
</tr>
</tbody>
</table>
Dan Boneh, Ben Lynn, and Hovav Shacham.
Short signatures from the weil pairing.

John Douceur.
The Sybil Attack.

Seth Gilbert and Nancy Lynch.
Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services.