Special protocols

Valtteri Niemi

(text also from Timo Karvi and Ari Renvall)

Department of Computer Science
University of Helsinki, Finland

Fall 2017
In a Secret Sharing Scheme a secret $s$ (e.g. a cryptographic key) is divided into a number of shares $s_i$, which are delivered to a set of parties $P = \{P_1, \ldots, P_n\}$.

The shares are constructed so that only some predetermined subsets of $P$ can recover the secret by pooling their shares.

In the following we consider a special type of a secret sharing scheme called a threshold scheme.
Let $t$ and $n$ be positive integers and $t < n$. A \textit{(t, n)-threshold scheme} is a method by which a trusted dealer $D$, knowing a secret value $s$, computes shares $s_i$ and delivers them to parties $P_i\ (i = 1, \ldots, n)$ in such a way that

- any set of $t$ parties can compute $s$ by pooling their shares; and
- no set of less than $t$ parties can compute $s$.

It is assumed that $D$ does not belong to $P$ and that the shares are distributed secretly so that no party knows the shares of other parties.
Assume that the secret $s$ is an element of $\mathbb{Z}_p$, where $p > n$ is a prime. In the Shamir Threshold Scheme the dealer $D$ performs the following steps:

1. $D$ selects $n$ distinct and non-zero elements $x_i \in \mathbb{Z}_p$. The element $x_i$ is publicly assigned to party $P_i$.
2. $D$ secretly selects $t-1$ random elements $a_i \in \mathbb{Z}_p$ ($i = 1, \ldots, t-1$).
3. For $1 \leq i \leq n$, $D$ computes $s_i = a(x_i)$, where $a(x)$ is the polynomial

   $$a(x) = s + \sum_{j=1}^{t-1} a_j x^j \pmod{p}.$$  

4. For $1 \leq i \leq n$, $D$ gives the share $s_i$ to $P_i$. 
The dealer $D$ with secret $s$ constructs a random polynomial $a(x)$ of degree (at most) $t - 1$ with constant term $a_0 = s$.

Each party knows the value of this polynomial at one point $(s_i = a(x_i))$, but nobody (except $D$) knows the polynomial itself.

Consider next that parties $P_{i_1}, \ldots, P_{i_t}$ want to determine $s$. Pooling their shares they know together $t$ points of $a(x)$. Therefore they obtain $t$ linear equations with $t$ unknowns $(s, a_1, \ldots, a_{t-1})$.

If these equations are linearly independent then the secret $s$ can be solved.
An example

Let $p = 17$, $t = 3$, $n = 5$, and assume that party $P_i$ is assigned the value $x_i = i$. Suppose that parties $P_1$, $P_3$ and $P_5$ pool their shares $s_1 = 8$, $s_3 = 10$ and $s_5 = 11$. The polynomial $a(x)$ is now

$$a(x) = s + a_1 x + a_2 x^2,$$

so the parties obtain equations

$$\begin{align*}
s + a_1 + a_2 &= 8 \pmod{17} \\
s + 3a_1 + 9a_2 &= 10 \pmod{17} \\
s + 5a_1 + 8a_2 &= 11 \pmod{17}
\end{align*}$$

This set of equations has the unique solution $s = 13$, $a_1 = 10$ and $a_2 = 2$. The secret is therefore 13.
When \( t \) parties pool their shares, the corresponding \( t \) equations need to be linearly independent.

In fact it can be shown that this is always the case.

Consider next that a group of less than \( t \), say \( t - 1 \), parties try to figure out the secret \( s \). They obtain \( t - 1 \) linear equations with \( t \) unknowns. By guessing the secret \( s \) they obtain \( t \)th equation \((a(0) = s)\).

Then they could find a unique solution.

Thus, for any possible value for the secret \( s \) there exists a unique polynomial \( a(x) \) that would have given them the same shares they have now. It means that any value for \( s \) is still equally probable.

The conclusion is that the \( t - 1 \) (or less) shares give no information on the secret \( s \).
When $t$ parties pool their shares, the corresponding $t$ equations need to be linearly independent.

In fact it can be shown that this is always the case.

Consider next that a group of less than $t$, say $t - 1$, parties try to figure out the secret $s$. They obtain $t - 1$ linear equations with $t$ unknowns. By guessing the secret $s$ they obtain $t$th equation $(a(0) = s)$

Then they could find a unique solution.

Thus, for any possible value for the secret $s$ there exists a unique polynomial $a(x)$ that would have given them the same shares they have now. It means that any value for $s$ is still equally probable.

The conclusion is that the $t - 1$ (or less) shares give no information on the secret $s$. 
When $t$ parties pool their shares, the corresponding $t$ equations need to be linearly independent.

In fact it can be shown that this is always the case.

Consider next that a group of less than $t$, say $t - 1$, parties try to figure out the secret $s$. They obtain $t - 1$ linear equations with $t$ unknowns. By guessing the secret $s$ they obtain $t$th equation ($a(0) = s$)

Then they could find a unique solution.

Thus, for any possible value for the secret $s$ there exists a unique polynomial $a(x)$ that would have given them the same shares they have now. It means that any value for $s$ is still equally probable.

The conclusion is that the $t - 1$ (or less) shares give no information on the secret $s$. 


When \( t \) parties pool their shares, the corresponding \( t \) equations need to be linearly independent.

In fact it can be shown that this is always the case.

Consider next that a group of less than \( t \), say \( t - 1 \), parties try to figure out the secret \( s \). They obtain \( t - 1 \) linear equations with \( t \) unknowns. By guessing the secret \( s \) they obtain \( t \)th equation \((a(0) = s)\)

Then they could find a unique solution.

Thus, for any possible value for the secret \( s \) there exists a unique polynomial \( a(x) \) that would have given them the same shares they have now. It means that any value for \( s \) is still equally probable.

The conclusion is that the \( t - 1 \) (or less) shares give no information on the secret \( s \).
When $t$ parties pool their shares, the corresponding $t$ equations need to be linearly independent.

In fact it can be shown that this is always the case.

Consider next that a group of less than $t$, say $t - 1$, parties try to figure out the secret $s$. They obtain $t - 1$ linear equations with $t$ unknowns. By guessing the secret $s$ they obtain $t$th equation ($a(0) = s$)

Then they could find a unique solution.

Thus, for any possible value for the secret $s$ there exists a unique polynomial $a(x)$ that would have given them the same shares they have now. It means that any value for $s$ is still equally probable.

The conclusion is that the $t - 1$ (or less) shares give no information on the secret $s$. 
When $t$ parties pool their shares, the corresponding $t$ equations need to be linearly independent.

In fact it can be shown that this is always the case.

Consider next that a group of less than $t$, say $t-1$, parties try to figure out the secret $s$. They obtain $t-1$ linear equations with $t$ unknowns. By guessing the secret $s$ they obtain $t$th equation $(a(0) = s)$

Then they could find a unique solution.

Thus, for any possible value for the secret $s$ there exists a unique polynomial $a(x)$ that would have given them the same shares they have now. It means that any value for $s$ is still equally probable.

The conclusion is that the $t-1$ (or less) shares give no information on the secret $s$. 
Cryptocurrency: Bitcoin

- A currency whose value is based on cryptography.
- No trusted authority.
- Instead, integrity of the system is maintained in peer-to-peer fashion.
- Transactions are digitally signed.
- All transactions are kept together in a hash chain.
ECDSA is used for digital signatures.

The curve is $y^2 = x^3 + 7$.

The base field is $\mathbb{Z}_p$ where $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$.

Individual bitcoins are assigned to public keys.

Whoever has the private key controls the bitcoin.

Transactions are statements of the form $X$ pays $Y$ bitcoins to $Z$.

Statement is signed by private key of $X$. 
All transactions are **broadcasted** and propagated to the whole network of bitcoin users.

- Logs are maintained in a **distributed database**.
- One or more transactions are collected to a single **block**.
- Blocks are maintained in a **block chain**.
- **Integrity** of the chain is maintained by computing a cryptographic **hash** over the data that includes:
  - hash of the previous block;
  - All transactions in the block;
  - A random **nonce**.
All transactions are **broadcasted** and propagated to the whole network of bitcoin users.

Logs are maintained in a **distributed database**.

One or more transactions are collected to a single **block**.

Blocks are maintained in a **block chain**.

**Integrity** of the chain is maintained by computing a **cryptographic hash** over the data that includes:

- **Hash** of the previous block;
- All transactions in the block;
- A random **nonce**.
All transactions are **broadcasted** and propagated to the whole network of bitcoin users.

Logs are maintained in a **distributed database**.

One or more transactions are collected to a single **block**.

Blocks are maintained in a **block chain**.

Integrity of the chain is maintained by computing a **cryptographic hash** over the data that includes:

- hash of the previous block;
- All transactions in the block;
- A random **nonce**.
All transactions are broadcasted and propagated to the whole network of bitcoin users.

Logs are maintained in a distributed database.

One or more transactions are collected to a single block.

Blocks are maintained in a block chain.

Integrity of the chain is maintained by computing a cryptographic hash over the data that includes:

- hash of the previous block;
- All transactions in the block;
- A random nonce.
Currency mining

- Anybody can compute the hash value for the new block.
- However, there is an extra requirement: if the hash value is interpreted as a number in binary form, then the value has to be smaller than a threshold value.
- Many different nonces have to be tried before a suitable hash is found. The transactions are confirmed after a suitable nonce has been found.
- Whoever comes up first with a suitable nonce gets new 25 bit coins. This is the only way to generate new currency to the system.
- The threshold becomes gradually smaller all the time. It is meant to be adjusted so that a new block is confirmed once in ten minutes (in average).
Currency mining

- Anybody can compute the hash value for the new block.
- However, there is an extra requirement: if the hash value is interpreted as a **number** in binary form, then the value has to be smaller than a **threshold** value.
- Many different nonces have to be tried before a suitable hash is found. The transactions are **confirmed** after a suitable nonce has been found.
- Whoever comes up first with a suitable nonce gets new 25 bit coins. This is the **only way** to generate new currency to the system.
- The threshold becomes gradually smaller all the time. It is meant to be adjusted so that a new block is confirmed once in **ten minutes** (in average).
Currency mining

- Anybody can compute the hash value for the new block.
- However, there is an extra requirement: if the hash value is interpreted as a number in binary form, then the value has to be smaller than a threshold value.
- Many different nonces have to be tried before a suitable hash is found. The transactions are confirmed after a suitable nonce has been found.
- Whoever comes up first with a suitable nonce gets new 25 bit coins. This is the only way to generate new currency to the system.
- The threshold becomes gradually smaller all the time. It is meant to be adjusted so that a new block is confirmed once in ten minutes (in average).
Currency mining

- Anybody can compute the hash value for the new block.
- However, there is an extra requirement: if the hash value is interpreted as a number in binary form, then the value has to be smaller than a threshold value.
- Many different nonces have to be tried before a suitable hash is found. The transactions are confirmed after a suitable nonce has been found.
- Whoever comes up first with a suitable nonce gets new 25 bit coins. This is the only way to generate new currency to the system.
- The threshold becomes gradually smaller all the time. It is meant to be adjusted so that a new block is confirmed once in ten minutes (in average).
Currency mining

- Anybody can compute the hash value for the new block.
- However, there is an extra requirement: if the hash value is interpreted as a number in binary form, then the value has to be smaller than a threshold value.
- Many different nonces have to be tried before a suitable hash is found. The transactions are confirmed after a suitable nonce has been found.
- Whoever comes up first with a suitable nonce gets new 25 bit coins. This is the only way to generate new currency to the system.
- The threshold becomes gradually smaller all the time. It is meant to be adjusted so that a new block is confirmed once in ten minutes (in average).
In an 1-2 oblivious transfer protocol a sender $S$ has two secrets $s_0$ and $s_1$. Exactly one of these secrets, say $s_b$ is to be delivered to $R$ (the receiver) in such a way that

- $R$ learns $s_b$ but does not learn anything about the other secret; and
- $S$ will not know which secret $R$ learned.

The following example gives a solution for this problem.
A protocol for 1-2 oblivious transfer.

1. $S$ sends to $R$ an RSA public key $(e, n)$ together with two randomly chosen values $x_0$ and $x_1$.

2. $R$ selects $b \in \{0, 1\}$ (depending on which secret $R$ wants) and a random key $k$. Then $R$ sends $v = x_b + k^e \pmod n$ to $S$.

3. $S$ computes (using the RSA private key) $k_0 = (v - x_0)^d \pmod n$ and $k_1 = (v - x_1)^d \pmod n$. Now $k_b = k$ but $S$ still does not get to know $b$. $S$ sends $y_0 = s_0 + k_0 \pmod n$ and $y_1 = s_1 + k_1 \pmod n$ to $R$.

4. $R$ computes $s_b = y_b - k \pmod n$. Note that $R$ has no clue about the other $k_i$ value and therefore cannot get any information about the other secret $s_i$.

This quite simple protocol seems to satisfy the requirements. $R$ obtains one of the secrets while only $R$ knows which one.
Secure multiparty computation is a protocol for a group of people whose goal is to compute the value of a function of many variables. Each member of the group provides one (or more) variables as inputs. The value of the function is to be known to everybody, but nobody should be able to learn other member’s inputs, other than what is obvious from the result.
**Yao’s garbled circuits**

**Simple example:** Suppose Bob chooses input bits to a logical gate while Alice chooses what kind of gate it is. Neither wants to reveal these selections to the other party.

![Diagram](image)

**Construction (Alice):**

- Assign random values to both 0 and 1 for every wire in the circuit
- Create an encrypted truth table, using *authenticated encryption*
- Permute the encrypted truth table to get the garbled truth table

<table>
<thead>
<tr>
<th></th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k^0_1$</td>
<td>$k^0_2$</td>
<td>$k^0_3$</td>
<td></td>
</tr>
<tr>
<td>$k^0_1$</td>
<td>$k^1_2$</td>
<td>$k^1_3$</td>
<td></td>
</tr>
<tr>
<td>$k^1_1$</td>
<td>$k^0_2$</td>
<td>$k^1_3$</td>
<td></td>
</tr>
<tr>
<td>$k^1_1$</td>
<td>$k^1_2$</td>
<td>$k^1_3$</td>
<td></td>
</tr>
</tbody>
</table>
Yao’s garbled circuits

Construction (Alice):

- Assign random values to both 0 and 1 for every wire in the circuit
- Create an encrypted truth table, using *authenticated encryption*
- Permute the encrypted truth table to get the garbled truth table
Yao’s garbled circuits

Construction (Alice):

- Assign random values to both 0 and 1 for every wire in the circuit
- Create an encrypted truth table, using **authenticated encryption**
- Permute the encrypted truth table to get the garbled truth table
Yao’s garbled circuits

Computation (Bob):

- To compute $g(0, 1)$: Using the 1-2 oblivious transfer protocol (with the help from Alice), Bob obtains random values $k_0^1$ and $k_1^2$ that correspond to the chosen input bits.
- Use keys $k_1^0$ and $k_2^1$ to decrypt the encrypted truth table. Only $\text{En}_{k_1^0}(\text{En}_{k_2^1}(k_3^1))$ can be decrypted successfully, resulting to $k_3^1$.
- Bob tells $k_3^1$ to Alice, who reveals that this value corresponds to 1 (which is correct since $g(0, 1) = 0 \lor 1 = 1$).

<table>
<thead>
<tr>
<th>Garbled truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{En}<em>{k_1^1}(\text{En}</em>{k_2^0}(k_3^1))$</td>
</tr>
<tr>
<td>$\text{En}<em>{k_1^0}(\text{En}</em>{k_2^1}(k_3^1))$</td>
</tr>
<tr>
<td>$\text{En}<em>{k_1^1}(\text{En}</em>{k_2^1}(k_3^1))$</td>
</tr>
<tr>
<td>$\text{En}<em>{k_1^0}(\text{En}</em>{k_2^0}(k_3^0))$</td>
</tr>
</tbody>
</table>
Yao’s garbled circuits

<table>
<thead>
<tr>
<th>Garbled truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{En}<em>{k_1^1}(\text{En}</em>{k_2^0}(k_3^1)) )</td>
</tr>
<tr>
<td>( \text{En}<em>{k_1^0}(\text{En}</em>{k_2^1}(k_3^1)) )</td>
</tr>
<tr>
<td>( \text{En}<em>{k_1^1}(\text{En}</em>{k_2^1}(k_3^1)) )</td>
</tr>
<tr>
<td>( \text{En}<em>{k_1^0}(\text{En}</em>{k_2^0}(k_3^0)) )</td>
</tr>
</tbody>
</table>

Computation (Bob):

- To compute \( g(0, 1) \): Using the 1-2 oblivious transfer protocol (with the help from Alice), Bob obtains random values \( k_0^1 \) and \( k_1^1 \) that correspond to the chosen input bits.

- Use keys \( k_1^0 \) and \( k_2^1 \) to decrypt the encrypted truth table. Only \( \text{En}_{k_1^0}(\text{En}_{k_2^1}(k_3^1)) \) can be decrypted successfully, resulting to \( k_3^1 \).

- Bob tells \( k_3^1 \) to Alice, who reveals that this value corresponds to 1 (which is correct since \( g(0, 1) = 0 \lor 1 = 1 \)).
Yao’s garbled circuits

<table>
<thead>
<tr>
<th>Garbled truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{En}<em>{k_1^1}(\text{En}</em>{k_2^0}(k_3^1))$</td>
</tr>
<tr>
<td>$\text{En}<em>{k_1^0}(\text{En}</em>{k_2^1}(k_3^1))$</td>
</tr>
<tr>
<td>$\text{En}<em>{k_1^1}(\text{En}</em>{k_2^1}(k_3^1))$</td>
</tr>
<tr>
<td>$\text{En}<em>{k_1^0}(\text{En}</em>{k_2^0}(k_3^0))$</td>
</tr>
</tbody>
</table>

Computation (Bob):

- To compute $g(0, 1)$: Using the 1-2 oblivious transfer protocol (with the help from Alice), Bob obtains random values $k_1^0$ and $k_2^1$ that correspond to the chosen input bits.

- Use keys $k_1^0$ and $k_2^1$ to decrypt the encrypted truth table. Only $\text{En}_{k_1^0}(\text{En}_{k_2^1}(k_3^1))$ can be decrypted successfully, resulting to $k_3^1$.

- Bob tells $k_3^1$ to Alice, who reveals that this value corresponds to 1 (which is correct since $g(0, 1) = 0 \lor 1 = 1$).
What did parties learn?

- What did Alice learn?
  - Answer: the output of OR-gate is 1, hence at least one of Bob’s input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.
- What did Bob learn?
  - Answer: Alice’s gate is such that 0 and 1 as inputs produce 1 as output.
  - At least, OR-gate or NAND-gate are possible!
What did parties learn?

- What did Alice learn?
  - Answer: the output of OR-gate is 1, hence at least one of Bob’s input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

- What did Bob learn?
  - Answer: Alice’s gate is such that 0 and 1 as inputs produce 1 as output.
  - At least, OR-gate or NAND-gate are possible!
What did parties learn?

- What did Alice learn?
  - Answer: the output of OR-gate is 1, hence at least one of Bob’s input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

- What did Bob learn?
  - Answer: Alice’s gate is such that 0 and 1 as inputs produce 1 as output.
  - At least, OR-gate or NAND-gate are possible!
What did parties learn?

- What did Alice learn?
  - Answer: the output of OR-gate is 1, hence at least one of Bob’s input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

- What did Bob learn?
  - Answer: Alice’s gate is such that 0 and 1 as inputs produce 1 as output.
  - At least, OR-gate or NAND-gate are possible!
What did parties learn?

What did Alice learn?
Answer: the output of OR-gate is 1, hence at least one of Bob’s input bits is also 1.
But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

What did Bob learn?
Answer: Alice’s gate is such that 0 and 1 as inputs produce 1 as output.
At least, OR-gate or NAND-gate are possible!
What did parties learn?

- What did Alice learn?
  Answer: the output of OR-gate is 1, hence at least one of Bob’s input bits is also 1.
  But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

- What did Bob learn?
  Answer: Alice’s gate is such that 0 and 1 as inputs produce 1 as output.
  At least, OR-gate or NAND-gate are possible!
Extending the example

- The output of one gate can directly be used as an input to another gate.
- This implies that the same model can be extended to more complex circuits.
- Summarizing: Bob controls input bits while Alice controls the function to be computed.
- i.e. Alice and Bob together compute \( y = f(x) \).
- Alice has chosen \( f \) while Bob has chosen \( x \).
- Alice does not learn anything more about \( x \) than what can be derived from \( y \).
- Bob does not learn anything more about \( f \) than what can be derived from \( y \) (except maybe number of gates in a circuit that computes \( f \)).
The output of one gate can directly be used as an input to another gate.

This implies that the same model can be extended more complex circuits.

Summarizing: Bob controls input bits while Alice controls the function to be computed.

i.e. Alice and Bob together compute $y = f(x)$.

Alice has chosen $f$ while Bob has chosen $x$.

Alice does not learn anything more about $x$ than what can be derived from $y$.

Bob does not learn anything more about $f$ than what can be derived from $y$ (except maybe number of gates in a circuit that computes $f$).
Extending the example

- The output of one gate can directly be used as an input to another gate.
- This implies that the same model can be extended more complex circuits.
- Summarizing: Bob controls input bits while Alice controls the function to be computed.
- i.e. Alice and Bob together compute \( y = f(x) \).
- Alice has chosen \( f \) while Bob has chosen \( x \).
- Alice does not learn anything more about \( x \) than what can be derived from \( y \).
- Bob does not learn anything more about \( f \) than what can be derived from \( y \) (except maybe number of gates in a circuit that computes \( f \)).
Extending the example

- The output of one gate can directly be used as an input to another gate.
- This implies that the same model can be extended more complex circuits.
- Summarizing: Bob controls input bits while Alice controls the function to be computed.
- i.e. Alice and Bob together compute \( y = f(x) \).
- Alice has chosen \( f \) while Bob has chosen \( x \).
- Alice does not learn anything more about \( x \) than what can be derived from \( y \).
- Bob does not learn anything more about \( f \) than what can be derived from \( y \) (except maybe number of gates in a circuit that computes \( f \)).
Extending the example

- The output of one gate can directly be used as an input to another gate.
- This implies that the same model can be extended more complex circuits.
- Summarizing: Bob controls input bits while Alice controls the function to be computed.
- i.e. Alice and Bob together compute $y = f(x)$.
- Alice has chosen $f$ while Bob has chosen $x$.
  - Alice does not learn anything more about $x$ than what can be derived from $y$.
  - Bob does not learn anything more about $f$ than what can be derived from $y$ (except maybe number of gates in a circuit that computes $f$).
Extending the example

- The output of one gate can directly be used as an input to another gate.
- This implies that the same model can be extended more complex circuits.
- Summarizing: Bob controls input bits while Alice controls the function to be computed.
- i.e. Alice and Bob together compute \( y = f(x) \).
- Alice has chosen \( f \) while Bob has chosen \( x \).
- Alice does not learn anything more about \( x \) than what can be derived from \( y \).
- Bob does not learn anything more about \( f \) than what can be derived from \( y \) (except maybe number of gates in a circuit that computes \( f \)).
Extending the example

- The output of one gate can directly be used as an input to another gate.
- This implies that the same model can be extended more complex circuits.
- Summarizing: Bob controls input bits while Alice controls the function to be computed.
- i.e. Alice and Bob together compute $y = f(x)$.
- Alice has chosen $f$ while Bob has chosen $x$.
- Alice does not learn anything more about $x$ than what can be derived from $y$.
- Bob does not learn anything more about $f$ than what can be derived from $y$ (except maybe number of gates in a circuit that computes $f$).
Let us assume Alice is a semi-honest trusted party who:
- controls the gate but has no interest in keeping the gate secret
- Bob and Carol provide the input bits to the OR-gate.
- Carol takes half of the burden from Bob:
  - (In the example above) Carol finds out $k^1_2$ by 1-2 oblivious transfer and Carol does decryption with $k^1_2$.
- If this order of input bits is used then it is Carol who gets the value $k^1_3$ and provides it to Alice.
Let us assume Alice is a semi-honest trusted party who:
- controls the gate but has no interest in keeping the gate secret
- Bob and Carol provide the input bits to the OR-gate.
- Carol takes half of the burden from Bob:
  - (In the example above) Carol finds out $k_2^1$ by 1-2 oblivious transfer and Carol does decryption with $k_2^1$.
  - If this order of input bits is used then it is Carol who gets the value $k_3^1$ and provides it to Alice.
Let us assume Alice is a semi-honest trusted party who:
- controls the gate but has no interest in keeping the gate secret
- Bob and Carol provide the input bits to the OR-gate.
- Carol takes half of the burden from Bob:
  - Carol finds out $k_1^1$ by 1-2 oblivious transfer and Carol does decryption with $k_2^1$.
  - If this order of input bits is used then it is Carol who gets the value $k_3^1$ and provides it to Alice.
Another extension

- Let us assume Alice is a semi-honest trusted party who:
  - controls the gate but has no interest in keeping the gate secret
  - Bob and Carol provide the input bits to the OR-gate.
  - Carol takes half of the burden from Bob:
    - (In the example above) Carol finds out $k_2^1$ by 1-2 oblivious transfer and Carol does decryption with $k_2^1$.
    - If this order of input bits is used then it is Carol who gets the value $k_3^1$ and provides it to Alice.
Another extension

Let us assume Alice is a semi-honest trusted party who:
- controls the gate but has no interest in keeping the gate secret
- Bob and Carol provide the input bits to the OR-gate.
- Carol takes half of the burden from Bob:
- (In the example above) Carol finds out $k_2^1$ by 1-2 oblivious transfer and Carol does decryption with $k_2^1$.
- If this order of input bits is used then it is Carol who gets the value $k_3^1$ and provides it to Alice.
Let us assume Alice is a **semi-honest trusted** party who:
- controls the gate but has no interest in keeping the gate secret
- Bob and Carol provide the input bits to the OR-gate.
- Carol takes **half** of the burden from Bob:
- (In the example above) Carol finds out $k_2^1$ by 1-2 oblivious transfer and Carol does decryption with $k_2^1$.
- If this order of input bits is used then it is Carol who gets the value $k_3^1$ and provides it to Alice.
What did parties learn?

- **What did Alice learn?**
  - Answer: the output of OR-gate is 1, hence at least one of the input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

- **What did Bob learn?**
  - Answer: Bob’s input bit is 0 while the output bit is 1. Hence, Bob learnt that Carol’s input bit must be 1.

- **What did Carol learn?**
  - Answer: Carol’s input bit is 1 while the output bit is 1. Hence, Carol did not learn whether Bob’s input bit is 0 or 1.
What did parties learn?

- What did Alice learn?
  - Answer: the output of OR-gate is 1, hence at least one of the input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

- What did Bob learn?
  - Answer: Bob’s input bit is 0 while the output bit is 1. Hence, Bob learnt that Carol’s input bit must be 1.

- What did Carol learn?
  - Answer: Carol’s input bit is 1 while the output bit is 1. Hence, Carol did not learn whether Bob’s input bit is 0 or 1.
What did parties learn?

- What did Alice learn?
  - Answer: the output of OR-gate is 1, hence at least one of the input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

- What did Bob learn?
  - Answer: Bob’s input bit is 0 while the output bit is 1. Hence, Bob learnt that Carol’s input bit must be 1.

- What did Carol learn?
  - Answer: Carol’s input bit is 1 while the output bit is 1. Hence, Carol did not learn whether Bob’s input bit is 0 or 1.
What did parties learn?

• What did Alice learn?
  Answer: the output of OR-gate is 1, hence at least one of the input bits is also 1.
  But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

• What did Bob learn?
  Answer: Bob’s input bit is 0 while the output bit is 1. Hence, Bob learnt that Carol’s input bit must be 1.

• What did Carol learn?
  Answer: Carol’s input bit is 1 while the output bit is 1. Hence, Carol did not learn whether Bob’s input bit is 0 or 1.
What did parties learn?

- What did Alice learn?
  - Answer: the output of OR-gate is 1, hence at least one of the input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

- What did Bob learn?
  - Answer: Bob’s input bit is 0 while the output bit is 1. Hence, Bob learnt that Carol’s input bit must be 1.

- What did Carol learn?
  - Answer: Carol’s input bit is 1 while the output bit is 1. Hence, Carol did not learn whether Bob’s input bit is 0 or 1.
What did parties learn?

- What did Alice learn?
  - Answer: the output of OR-gate is 1, hence at least one of the input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.
- What did Bob learn?
  - Answer: Bob’s input bit is 0 while the output bit is 1. Hence, Bob learnt that Carol’s input bit must be 1.
- What did Carol learn?
  - Answer: Carol’s input bit is 1 while the output bit is 1. Hence, Carol did not learn whether Bob’s input bit is 0 or 1.
What did parties learn?

- **What did Alice learn?**
  - Answer: the output of OR-gate is 1, hence at least one of the input bits is also 1.
  - But Alice did not learn which of the bits is 1 and whether maybe both bits are 1.

- **What did Bob learn?**
  - Answer: Bob’s input bit is 0 while the output bit is 1. Hence, Bob learnt that Carol’s input bit must be 1.

- **What did Carol learn?**
  - Answer: Carol’s input bit is 1 while the output bit is 1. Hence, Carol did not learn whether Bob’s input bit is 0 or 1.
Authenticated encryption

Often securing data requires protecting both integrity and confidentiality. Thus, we need message authentication codes and encryption. But in which order we should apply these two mechanisms?
All of the following seem to be feasible options:

- **MAC-then-encrypt:**
  First a MAC is computed over the plaintext with integrity key $K_{int}$; then MAC is appended to the end of the plaintext and the result is encrypted with encryption key $K_{enc}$.

- **encrypt-and-MAC:**
  In parallel, we encrypt the plaintext with $K_{enc}$, and compute MAC over the plaintext, using $K_{int}$. Then results of the two parallel operations are finally concatenated.

- **encrypt-then-MAC:**
  First we encrypt the plaintext with $K_{enc}$, then a MAC is computed over the resulting ciphertext, using $K_{int}$. Finally, the MAC is appended to the ciphertext.
All of the following seem to be feasible options:

- **MAC-then-encrypt:**
  First a MAC is computed over the plaintext with integrity key $K_{int}$; then MAC is appended to the end of the plaintext and the result is encrypted with encryption key $K_{enc}$.

- **encrypt-and-MAC:**
  In parallel, we encrypt the plaintext with $K_{enc}$, and compute MAC over the plaintext, using $K_{int}$. Then results of the two parallel operations are finally concatenated.

- **encrypt-then-MAC:**
  First we encrypt the plaintext with $K_{enc}$, then a MAC is computed over the resulting ciphertext, using $K_{int}$. Finally, the MAC is appended to the ciphertext.
All of the following seem to be feasible options:

- **MAC-then-encrypt:**
  First a MAC is computed over the plaintext with integrity key $K_{int}$; then MAC is appended to the end of the plaintext and the result is encrypted with encryption key $K_{enc}$.

- **encrypt-and-MAC:**
  In parallel, we encrypt the plaintext with $K_{enc}$, and compute MAC over the plaintext, using $K_{int}$. Then results of the two parallel operations are finally concatenated.

- **encrypt-then-MAC:**
  First we encrypt the plaintext with $K_{enc}$, then a MAC is computed over the resulting ciphertext, using $K_{int}$. Finally, the MAC is appended to the ciphertext.
All of the following seem to be feasible options:

- **MAC-then-encrypt:**
  First a MAC is computed over the plaintext with integrity key $K_{int}$; then MAC is appended to the end of the plaintext and the result is encrypted with encryption key $K_{enc}$.

- **encrypt-and-MAC:**
  In parallel, we encrypt the plaintext with $K_{enc}$, and compute MAC over the plaintext, using $K_{int}$. Then results of the two parallel operations are finally concatenated.

- **encrypt-then-MAC:**
  First we encrypt the plaintext with $K_{enc}$, then a MAC is computed over the resulting ciphertext, using $K_{int}$. Finally, the MAC is appended to the ciphertext.
All three options have also been used widely in practice. Arguably the three most popular communication security protocols have chosen a different option each:

- **SSL** applies MAC-then-encrypt;
- **SSH** applies encrypt-and-MAC;
- **IPsec** applies encrypt-then-MAC.
All three options have also been used widely in practice. Arguably the three most popular communication security protocols have chosen a different option each:

- **SSL** applies MAC-then-encrypt;
- **SSH** applies encrypt-and-MAC;
- **IPsec** applies encrypt-then-MAC.
All three options have also been used widely in practice. Arguably the three most popular communication security protocols have chosen a different option each:

- **SSL** applies MAC-then-encrypt;
- **SSH** applies encrypt-and-MAC;
- **IPsec** applies encrypt-then-MAC.
All three options have also been used widely in practice. Arguably the three most popular communication security protocols have chosen a different option each:

- **SSL** applies MAC-then-encrypt;
- **SSH** applies encrypt-and-MAC;
- **IPsec** applies encrypt-then-MAC.
All three options are secure as such but each option has also shortcomings:

- **MAC-then-encrypt:**
  An attacker could modify the ciphertext but decryption would still work.
  There is only a tiny chance that the decrypted plaintext and the decrypted MAC would match.
  However, integrity of the ciphertext is not protected.

- **encrypt-and-MAC:**
  In addition to the above, the following scenario is possible:
  A replay protection mechanism is typically used, making encryption of the same plaintext different each time.
  Assuming nothing similar is applied for MAC, it is possible to notice whether the plaintext is a repetition of an earlier plaintext (why?).
Shortcomings

All three options are secure as such but each option has also shortcomings:

- **MAC-then-encrypt:**
  An attacker could modify the ciphertext but decryption would still work.
  There is only a tiny chance that the decrypted plaintext and the decrypted MAC would match.
  However, integrity of the ciphertext is not protected.

- **encrypt-and-MAC:**
  In addition to the above, the following scenario is possible:
  A replay protection mechanism is typically used, making encryption of the same plaintext different each time.
  Assuming nothing similar is applied for MAC, it is possible to notice whether the plaintext is a repetition of an earlier plaintext (why?).
All three options are secure as such but each option has also shortcomings:

- **MAC-then-encrypt:**
  An attacker could modify the ciphertext but decryption would still work.
  There is only a tiny chance that the decrypted plaintext and the decrypted MAC would match.
  However, integrity of the ciphertext is not protected.

- **encrypt-and-MAC:**
  In addition to the above, the following scenario is possible:
  A replay protection mechanism is typically used, making encryption of the same plaintext different each time.
  Assuming nothing similar is applied for MAC, it is possible to notice whether the plaintext is a repetition of an earlier plaintext (why?).
encrypt-then-MAC:
This option protects integrity of the ciphertext, but an attacker may still be able to do (at least an uncontrolled) modification to the plaintext:
An attacker may change the identifier for the encryption algorithm unless this parameter is also protected by MAC.
Similarly, the attacker may be able to change the initialization vector $IV$ (in case this parameter is needed for the encryption algorithm).
These kind of changes would cause a change in the plaintext although the key $K_{enc}$ is correct.
How about protecting both integrity and confidentiality at the same time?

Because we need to ensure both protection goals anyway, maybe we could construct a cryptographic primitive that provides both "at one go",

i.e. we look for an algorithm that processes the data only once but could still protect both integrity and confidentiality.

This means we could improve both security and efficiency.
Another approach to AE

- How about protecting both integrity and confidentiality at the same time?
- Because we need to ensure both protection goals anyway, maybe we could construct a cryptographic primitive that provides both “at one go”,
- i.e. we look for an algorithm that processes the data only once but could still protect both integrity and confidentiality.
- This means we could improve both security and efficiency.
Another approach to AE

- How about protecting both integrity and confidentiality at the same time?
- Because we need to ensure both protection goals anyway, maybe we could construct a cryptographic primitive that provides both “at one go”,
- i.e. we look for an algorithm that processes the data only once but could still protect both integrity and confidentiality.
- This means we could improve both security and efficiency.
Another approach to AE

- How about protecting both integrity and confidentiality at the same time?
- Because we need to ensure both protection goals anyway, maybe we could construct a cryptographic primitive that provides both ”at one go” ,
- i.e. we look for an algorithm that processes the data only once but could still protect both integrity and confidentiality.
- This means we could improve both security and efficiency.
CBC for encryption

\[ \text{IV} \rightarrow E \rightarrow C_1 \]
\[ k \rightarrow E \rightarrow C_2 \]
\[ k \rightarrow E \rightarrow C_3 \]
The two modes look very similar...
...but combining them to one mode for authenticated encryption is not trivial.
The two modes look very similar...

...but combining them to one mode for authenticated encryption is not trivial.
The two modes look very similar...

...but combining them to one mode for authenticated encryption is not trivial.
One solution is called **Integrity-Aware CBC mode**.

But how are the **whitening** values $S_i$ generated? These need to be different random strings.
One solution is called **Integrity-Aware CBC mode**.

But how are the **whitening** values $S_i$ generated? These need to be different random strings.
Fortunately, it is sufficient for security that the values $S_i$ are pairwise independent. They do not have to be completely independent from each other.

Let $i + 1 = a_0 + a_1 2 + a_2 2^2 + \ldots + a_k 2^k$.

Now $S_i = a_0 R_0 \oplus a_1 R_1 \oplus a_2 R_2 \oplus \ldots \oplus a_k R_k$. 
Fortunately, it is sufficient for security that the values $S_i$ are pairwise independent. They do not have to be completely independent from each other.

Let $i + 1 = a_0 + a_1 2 + a_2 2^2 + \ldots + a_k 2^k$.

Now $S_i = a_0 R_0 \oplus a_1 R_1 \oplus a_2 R_2 \oplus \ldots \oplus a_k R_k$. 
Fortunately, it is sufficient for security that the values $S_i$ are pairwise independent. They do not have to be completely independent from each other.

Let $i + 1 = a_0 + a_1 2 + a_2 2^2 + \ldots + a_k 2^k$.

Now $S_i = a_0 R_0 \oplus a_1 R_1 \oplus a_2 R_2 \oplus \ldots \oplus a_k R_k$. 
- $n + 2$ one-block encryptions are needed for processing $P_1, \ldots, P_n$ together with $IV$ and the checksum (over $P_1, \ldots, P_n$).
- In addition, $\log n$ one-block encryptions are needed for generation of $R_1, \ldots, R_k$.
- Rest of operations are simple XOR operations.
- This is to be compared with (around) $2n$ one-block encryptions needed for all three traditional modes: MAC-then-encrypt, encrypt-and-MAC, encrypt-then-MAC.
IACBC complexity

- $n + 2$ one-block encryptions are needed for processing $P_1, \ldots, P_n$ together with $IV$ and the checksum (over $P_1, \ldots, P_n$).
- In addition, $\log n$ one-block encryptions are needed for generation of $R_1, \ldots, R_k$.
- Rest of operations are simple XOR operations.
- This is to be compared with (around) $2n$ one-block encryptions needed for all three traditional modes: MAC-then-encrypt, encrypt-and-MAC, encrypt-then-MAC.
$n + 2$ one-block encryptions are needed for processing $P_1, \ldots, P_n$ together with IV and the checksum (over $P_1, \ldots, P_n$).

In addition, $\log n$ one-block encryptions are needed for generation of $R_1, \ldots, R_k$.

Rest of operations are simple XOR operations.

This is to be compared with (around) $2n$ one-block encryptions needed for all three traditional modes: MAC-then-encrypt, encrypt-and-MAC, encrypt-then-MAC.
$n + 2$ one-block encryptions are needed for processing $P_1, \ldots, P_n$ together with $IV$ and the checksum (over $P_1, \ldots, P_n$).

In addition, $\log n$ one-block encryptions are needed for generation of $R_1, \ldots, R_k$.

Rest of operations are simple XOR operations.

This is to be compared with (around) $2n$ one-block encryptions needed for all three traditional modes: MAC-then-encrypt, encrypt-and-MAC, encrypt-then-MAC.
ECB mode

\[
\begin{align*}
P_0 & \rightarrow E & P_1 & \rightarrow E & P_2 & \rightarrow E & \cdots & P_n & \rightarrow E \\
\downarrow & & \downarrow & & \downarrow & & \cdots & \downarrow \\
C_0 & & C_1 & & C_2 & & \cdots & C_n
\end{align*}
\]
Another solution for authenticated encryption is Offset Code Book mode.

Complexity of this parallelizable mode: \( n + 2 \) one-block encryptions.