1) Protocol I: $K_{AB} = t_b^x y_b^{r_a} = t_b^{x_A} (g^{x_b})^{r_a} = t_b^{x_A} (g^{r_a})^{x_b} = t_b^{x_A} t_A^{x_b}$.

If an attacker is able to break into secrets $x_A$ and $x_B$ and has recorded exchanged values $t_b$ and $t_A$, then the attacker is able to (reversely) compute the key $K_{AB}$. Hence, no forward secrecy.

Protocol II: $A \rightarrow B: (y_B)^{x_A} = (g^{x_b})^{x_A} = (g^{r_a})^{x_b}$

$B \rightarrow A: (y_A)^{x_B} = (g^{r_b})^{x_A}$

$K_{AB} = g^{r_a r_b}$. Hence, protocol II is the "normal" DH exchange with the additional protection that $g^{r_a}$ and $g^{r_b}$ are paired to long-term keys.

If an attacker gets $x_A$ and $x_B$, he can strip away the additional protection but "normal" DH still stands. Hence, protocol II has forward secrecy.

2) $x \mapsto x^2$ is bijective in $\mathbb{Z}_n$ if $p \equiv 3 (4)$.

This may be tested by choosing $y \equiv x^2 \pmod{n}$. If $n = pq$ now $P$ is asked to give back a square root of $y$. There is a good chance that this square root is not $x$ nor $-x$. Then $V$ can find factors of $n$ (no zero-knowledge).
3) We use the following "unorthodox" gates: (II)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a &gt; b</th>
<th>a = b</th>
<th>a ≥ b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Alice's number is greater than Bob's number if \( a_1 > b_1 \), smaller if \( a_1 < b_1 \). If \( a_1 = b_1 \) then we proceed to comparison of \( a_2, a_2, \ldots \) etc.

Thus the following circuit returns 1 if and only if Alice's number is at least as great as Bob's number: \( (a_1 > b_1) \) or \( (a_1 = b_1) \) and \( (a_2 > b_2) \) or \( (a_2 = b_2) \) and \( (a_3 > b_3) \)

A gambled circuit is used. Alice prepares all permuted and gambled truth tables. Bob obtains by 1-2 oblivious transfer random values corresponding to \( b_1, b_2, b_3 \). Alice does first decryptions for gates where \( a_1, a_2, a_3 \) are inputs. Rest of decryption are done by Bob. In the end, Alice reveals whether final value corresponds to 0 or 1.
(4) \( n = 143 = 13 \cdot 11 \) (but this is kept as secret)

(1) \( S \rightarrow R: \quad n = 143 \)
(2) \( R \rightarrow S: \quad x^2 = 625 \equiv 53 \pmod{143} \)

\[ R \text{ chooses } x = 25 \]

(3) \( S \) computes: \( a \mod 11 = 9 \) squares are \( \pm 3 \)
\[ a \mod 13 = 1 \quad \rightarrow 4 \quad \pm 1 \]

One possibility fulfilling the latter is \( x' = 13k + 1 \)

\[ x' \equiv 2k + 1 \pmod{11} \]

On the other hand, one possibility fulfilling the former is:

\[ 2k + 1 \equiv x' \equiv 3 \pmod{11} \]

\[ 2k \equiv 2 \pmod{11} \]

\[ k \equiv 1 \pmod{11} \]

\[ x' = 13 \cdot 1 + 1 = 14 \]

(4) \( S \rightarrow R: \quad y = 14 \).

\( R \) computes: \( 14 \neq 25, 14 \neq -25 \pmod{143} \)

\[ \gcd(x + y, n) = \gcd(25 + 14, 143) = \gcd(39, 143) \]

\[ 143 = 3 \cdot 39 + 26 \]
\[ 39 = 1 \cdot 26 + 13 \]
\[ 26 = 2 \cdot 13 \quad \therefore \gcd = 13 \]

In this run, \( R \) finds out a factor of 143.

If \( S \) chooses differently in phase (3), it may be that \( R \) gets 25 or -25 back and then does not learn factors of \( n \).
5) CBC-MAC:

```
\[ P_0 \quad P_1 \quad P_2 \quad P_{n-2} \quad P_{n-1} \]
\[ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \]
\[ E \quad E \quad E \quad E \quad E \quad E \]
```

CFB-MAC:

```
\[ P_0 \]
\[ \downarrow \]
\[ E \]
\[ \downarrow \]
\[ \beta_0 = P_0 \]
```

Comparing the figures, we see that \( \alpha \)'s and \( \beta \)'s matches:

* In both cases \( \text{MAC} = E_k(\beta_{n-1}) \)

\[
\alpha_{i+1} = E_k(\beta_i) \\
\beta_{i+1} = \alpha_{i+1} \oplus P_{i+1} = E_k(\beta_i) \oplus P_{i+1} \\
\alpha_n = \text{MAC}, \quad \beta_0 = P_0
\]

6) Receiving side:

(i) First compute \( R_i \) values:

\[ R_i \]

(ii) For each \( i = 1, \ldots, n \):

\[ S_i = a_0 R_0 \oplus a_1 R_1 \oplus \cdots \oplus a_n R_n \]

Can be parallelized.

(iii) Now decrypt:

\[ C_i \]

\[ S_i = S_i-1 \]

Checksum:

\[ c_i \]

With \( P_1, \ldots, P_n \): complete work on:

\[ \text{decrypt} \rightarrow \text{checksum} \rightarrow \text{MAC} \]