The personal number used is 199603177792.

1 Running the assignment parts

The assignments are all in different, clearly named Haskell files. To run them all, use `cabal run` or run the `Main.hs` file using GHC. This prints the output of all the separate assignment files (in the case of the first assignment, runs the supplied tests). There is also a compiled binary, called `Main`, for Linux x86-64 which can be run directly.

2 A Math Library for Cryptography

There is nothing out of the ordinary here. The extended Euclidian algorithm follows the textbook implementation. The Euler ϕ function is implemented naively, simply trying all integers below the input to see if the gcd is 1 or not. The modular inverse uses the EEA to find the coefficients for Bezout's identity. Fermat's primality test is also straightforward, but first of all tries to invalidate a prime by using Fermat's theorem, and only then checks if any of the tested numbers was a divisor.[1] The has collision probability is calculated with high precision, by checking the probability of getting a collision at each sample point.

The library also includes some functions that were helpful in the other assigments, especially `modN`, which uses exponentiation by squaring, something that became necessary to solve the ElGamal assignment in reasonable time.

3 Special Soundness of Fiat-Shamir sigma-protocol

Knowing that the same nonce was used once for a challenge $c = 0$, and once for $c = 1$, we can deduce the secret key.

Call the repeated nonce r. In the first step of the protocol, $R = r^2$ is sent to the verifier. We thus find as situation when R is repeated, and know that r was also repeated, with high probability. In one of the runs, the challenge sent by the verifier was $c = 0$, meaning the prover sent back r. Thus, we know r; it is s, the final message of the run. We turn to the run of the protocol that used the same R, and $c = 1$. In this case, the final message $s = rx$, x being the secret key. Since we know r, we can easily compute r^{-1} using our *modinv* function, and thus get $x = sr^{-1}$.

The function *collectInfo* gives back the two runs that used the same nonce. *recoverX* then calculates the secret key, x.

Decoded message: A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools.

[1] We did find searching up to $\frac{n}{3}$ a bit curious. Why not stop at roughly $\sqrt{n}$, e.g., by multiplying the tested number by itself each time and comparing to n?

4 Decrypting CBC with simple XOR

The weakness in this particular cipher is the simplistic use of the key. By using that me know the first message block, $m_0 = 199603177792$, that $IV = 6725DD9E6DE0$, and that $c_0 = k \oplus m_0 \oplus IV = 823C1EE8E02D$, we can easily calcuclate $k = c_0 \oplus m_0 \oplus IV$. We then use the key to decrypt the rest of the message, using a simple decryption circuit.

Decoded message: 199603177792Do or do not. There is not try. - Master Yoda000

As you can see, the first block is the one that was previously known, and the last block contains padding.

5 Attacking RSA

Since the same message has been encrypted by three different recipients and modulus we can solve c the linear congruence using the chinese remainder theorem:
$c = c1 mod N1$
$c = c2 mod N2$
$c = c3 mod N3$

Since we know that all recipients used the same public key, 3, we know that $c = m^3$. Therefore we can recover the message simply by taking the cube root of c.

Decoded message: Taher ElGamal

6 Attacking ElGamal

Since we have a limit key space due to the weak random number generator, i.e. 1000 different keys, we can do an exhaustive search to find the used private key. By comparing c1 with g^k and when $c1 == g^k$ k is the private key.

Next we calculate the inverse of the public key s.t we have $g^- x$ using our $modInv$ function from CryptoLib, which returns the Bèzout coefficients.

With access to the private key k and public key inverse we can now compute the shared secret which is $g^- xk$. Mulitple c2 with the shared secret we recover the the message.

Decoded message: The only fully secure symmetric cryptosystem is Bruce Schneier looking in a mirror.S