Solution Sheet 11

Cryptography and Security 2022

Solution 1 Distribution of Birthdays

1. It is $1/N$ due to uniform distribution.
2. It is again $1/N$ due to uniform distribution and independence.
3. The probability to be in a 3-collisions is $\binom{M-1}{2}\frac{1}{N^2}\left(1-\frac{1}{N}\right)^{M-3} \approx 1.9\%$.

   We could do the same computation for a 2-collision: the probability that exactly one other student share his birthday is $\binom{M-1}{1}\frac{1}{N}\left(1-\frac{1}{N}\right)^{M-2} \approx 18\%$.

   The probability that no student share his birthday is $\left(1-\frac{1}{N}\right)^{M-1} \approx 80\%$.
4. We have $\binom{M}{2}$ pairs of students. Each pair share the same birthday with probability $1/N$. So, is should be $\binom{M}{2}\frac{1}{N} \approx 8.9$.

   For triplets, this is $\binom{M}{3}\frac{1}{N^2} \approx 0.6$.
5. The pairs are from the pool of 2-collisions and 3-collisions. In a 3-collision, we have 3 pairs. In a 2-collision, we have a single pair. So, we have $3 \times 3 + 6 \times 1 = 15$ pairs of students with the same birthday. There is a gap between 15 and 8.9.

   Clearly, we have only 3 triplets of students sharing the same birthday. There is a gap between 3 and 0.6.

   Most probably, the distribution of birthdays in the class is not uniform. This can explain the discrepancy.
6. We have $\Pr[\text{collision}] \leq \binom{M}{2}\frac{1}{N}$ with $N = 10^\ell$. We can solve $\binom{M}{2}\frac{1}{10^\ell} \leq 0.01$ and obtain $\ell \geq 2 + \log_{10}\binom{M}{2} \approx 5.5$. So, $\ell = 6$ digits are enough for $\Pr[\text{collision}] \leq 1\%$.

   With $\ell = 5$ digits, we have

$$\Pr[\text{collision}] = 1 - \prod_{i=0}^{M-1}\left(1-\frac{i}{N}\right) \approx 0.03$$

   so 5 digits are not enough. Note that we obtain the same result by using the approximation

$$\Pr[\text{collision}] \approx 1 - e^{-\frac{M^2}{2N}}$$

Solution 2 RSA for Paranoids

1. The bottleneck is making an s-bit prime number, which can be done in $\mathcal{O}(s^4)$.

2. For plain RSA, the condition would be $\gcd(e,(p-1)(q-1))=1$. Here q is not a prime, and its factorization is not even known, so that computing $\varphi(pq)$ is not an easy task. We can guess that the condition we are looking for is $\gcd(e,p-1)=1$. We now show that this is sufficient to make E injective. For any $m_1, m_2 \in \{0,1,\ldots,2^{s-1}-1\}$, we have

$$\begin{aligned} E(m_1)=E(m_2) &\Rightarrow m_1^e \equiv m_2^e \pmod{pq} \\ &\Rightarrow m_1^e \equiv m_2^e \pmod{p}. \end{aligned}$$

As $\gcd(e,p-1)=1$, we can find (using the Extended Euclid Algorithm) two integers u,v such that $ue-v(p-1)=1$. Therefore

$$\begin{aligned} E(m_1)=E(m_2) &\Rightarrow m_1^{ue} \equiv m_2^{ue} \pmod{p} \\ &\Rightarrow m_1^{1+v(p-1)} \equiv m_2^{1+v(p-1)} \pmod{p} \\ &\Rightarrow m_1 \equiv m_2 \pmod{p}\ , \end{aligned}$$

using Fermat's Little Theorem. Finally, as $m_1 < p$ and $m_2 < p$, the last condition is sufficient to show that $m_1 = m_2$.

3. As $\gcd(e,p-1)=1$, we can compute $d = e^{-1} \bmod (p-1)$, so that there exists some $k \in \mathbf{Z}$ such that $ed = 1 + k(p-1)$. To decrypt, we compute

$$E(m)^d \bmod p = m^{ed} \bmod p = m^{1+k(p-1)} \bmod p = m\ ,$$

using Fermat's Little Theorem.

4. Encryption is a modular exponentiation, so that the complexity is $\mathcal{O}(s^3t^2)$ (exponent is of length s, multiplication of st-bit long integers is quadratic). Similarly, decryption's complexity is $\mathcal{O}(s^3)$ (integers are s-bit long). We can accept $\mathcal{O}(s^3)$ for both complexities if t is considered as a constant. The complexity of the key generation is the same as for plain RSA, that is, $\mathcal{O}(s^4)$ (prime generation).

5. If e is smaller than t, $m^e \bmod n$ is simply m^e, since $m^e < n$. So, anyone can extract eth roots over $\mathbf{Z}$ and decrypt the ciphertext c.

6. Clearly, the knowledge of p enables to compute the secret key and thus to decrypt. Conversely, suppose we can decrypt, i.e., we have access to a decryption machine that takes as an input any ciphertext and returns as an output the result of the decryption process on the ciphertext. We choose to submit the encryption of a *large* plaintext m such that $p < m < 2p$ (even if p is not known yet, such a m can be chosen as we know the size of p). We can write m as $m = p + u$, where $u < p$. The decryption machine allows to recover u easily. Indeed, if we submit the ciphertext $m^e \bmod n$ the decryption machine returns

$$m^{ed} \bmod p = (p+u)^{ed} \bmod p = u^{ed} \bmod p = u\ ,$$

as u was chosen smaller than p. Knowing m and u allows to recover p as $p = m - u$. Note that the same kind of ideas can be applied to the Rabin cryptosystem.

7. This works as in Question 6, since we have a decryption oracle at disposal.

8. To thwart this attack, one can add some redundancy in the message before encryption, and check the redundancy after decryption before disclosing the result.