CNS 24 HW1

b11902008 笪瑜庭

Classmates with whom I have discussed this homework: 洪銘德、官毓韋、蔡朝暐、陳愷欣、賴昭勳

---

1. CIA

- **confidentiality**: protecting from unauthorized disclosure
  - e.g. a login service should encrypt the communication between the user and itself with HTTPS, preventing adversaries from eavesdropping on passwords or other private information.
- **integrity**: protecting from unauthorised changes
  - e.g. if someone want to change her password, the old password is required to verify her identity.
- **availability**: ensuring intended users can access service
  - e.g. the login service has efficient load-balancing mechanism, remaining accessible during high traffic period or even a DDoS attack.

2. Hash Function

Let $H : X \to Y$ represent a hash function.

- **one-wayness**: Given y, it's computationally infeasible to find x such that $y = H(x)$
  - e.g. POW requires one-wayness to makes sure the prover expend a certain amount of computational effort.
- **weak collision resistance**: Given x, it's computationally infeasible to find $x' \neq x$ such that $H(x') = H(x)$
  - e.g. file integrity check with sha-256 only need weak collision resistance.
- **strong collision resistance**: It's computationally infeasible to find x and x' such that $x' \neq x$ and $H(x') = H(x)$
  - e.g. a database stores data in unique hashes, and a user can look up an arbitrary data by computing the hash of it.

3. Asymmetric Cryptography

3.1. An introduction to elliptic curves in cryptography

a)

REF: https://en.wikipedia.org/wiki/Elliptic_curve;
https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication#Point_addition;
https://crypto.stanford.edu/pbc/notes/elliptic/explicit.html;
https://web.math.ucsb.edu/~padraic/ucsb_2014_15/ccs_discrete_f2014/ccs_discrete_f2014_lecture9.pdf; *Elliptic Curves: Number Theory and Cryptography, Second Edition*; https://edu-ctf.csie.org/static/files/Crypto2_d829e930536f6a3b.pdf

i. Point Addition:

First, for the case where $Q = -P$, the addition is defined as $P + Q = P + (-P) = O$.
Second, for the case where $P \neq Q$, we can find the point $S = -(P + Q)$ resulting from the intersection of the elliptic curve $E : y^2 = x^3 + ax + b$ and the straight line $L : y = \lambda x + d$ defined by P and Q. The slope of the line is given by: $\lambda = \frac{y_Q - y_P}{x_Q - x_P}$. To find the coordinates of S:

$$\begin{aligned} &(\lambda x + d)^2 = x^3 + ax + b \\ \implies\ &x^3 - \lambda^2 x_2 - 2\lambda d x + ax + b - d^2 = 0 \end{aligned}$$

Since x_P, x_Q, and x_R are solutions, this equation has its roots at exactly the same x values as

$$\begin{aligned} &(x - x_P)(x - x_Q)(x - x_S) \\ &= x^3 + (-x_P - x_Q - x_S)x^2 + (x_P x_Q + x_P x_S + x_Q x_S)x - x_P x_Q x_S \end{aligned}$$

Then by equating the coefficients of x^2, we can solve $x_S = \lambda^2 - x_P - x_Q$, and by the slope of L, $y_S = \lambda(x_S - x_P) + y_P$
The coordinates of $R = P + Q = -S$ are therefore:

$$\begin{aligned} x_R &= \lambda^2 - x_P - x_Q \\ y_R &= \lambda(x_P - x_R) - y_P \end{aligned}$$

ii. Point Doubling:

Doubling a point P, which on the elliptic curve is the same as $P + Q$ where $Q = P$. For the case where $yP = 0$, $2P = P + P = O$; for the case where $y_P \neq= 0$, by using the point addition formulae, we calculate the slope of the tangent at the point (x_P, y_P) by $\frac{dE/dx}{dE/dy}$:

$$\lambda' = \frac{3x_P^2 + a}{2y_P}$$

The coordinates of the point $R = P + P = 2P$ is the same as the previous part, only with different slope λ':

$$\begin{aligned} x_R &= (\lambda')^2 - 2x_P \\ y_R &= \lambda'(x_P - x_R) - y_P \end{aligned}$$

b)

We want to compute $n \times P$, where n is a positive integer and P is a point on the elliptic curve. The key idea of exponentiation by squaring is to divide the exponent b into smaller parts (usually in binary representation) and compute intermediate results. Similarly, we can express n in binary form: $n = b_k b_{k-1} \ldots b_1 b_0$.

1. Initialize an accumulator R to the point at infinity, i.e. identity element O.
2. For each bit b_i in the binary representation of n:
   i. $R = 2R$ (point doubling).
   i. If $b_i = 1$, $R = R + P$.
   Repeat until all bits are processed.
3. In the end, $R = n \times P$.

If the binary representation of n has k bits, the number of iterations is k and the total number of additions (including point doubling) is of the order $O(k)$. Since $k = \log_2 n$, there are $O(\log n)$ additions.

c)

If the line passing P and Q does not intersect the elliptic curve at another point, then the line must be a tangent to the curve at one of the points. WLOG, suppose the line is tangent at Q. The addition is defined as $R = P + Q = -Q$. This definition does not break the group laws.

d)

Yes, discrete log problems on singular elliptic curves are easier to solve. This problem can be split into to 2 cases: i. the curve has a triple root, and ii. the curve has a double root.

In the book *Elliptic Curves: Number Theory and Cryptography, Second Edition* by Lawrence C. Washington, there are 2 useful theorems:

> **THEOREM 2.30**
> Let E be the curve y^2 = x^3 and let $Ens(K)$ be the nonsingular points on this curve with coordinates in K ,including the point $\infty = (0:1:0)$. The map
>
> $$E_{ns}(K) \to K,\ (x,y) \mapsto x/y,\ \infty \mapsto 0$$
>
> is a group isomorphism between $E_{ns}(K)$ and K, regarded as an additive group.

> **THEOREM 2.31**
> Let E be the curve $y^2 = x^2(x+a)$ with $0 \neq a \in K$. Let $E_{ns}(K)$ be the nonsingular points on E with coordinates in K. Let $\alpha^2 = a$. Consider the map
>
> $$\psi : (x,y) \mapsto \frac{y+\alpha x}{y-\alpha x},\ \infty \mapsto 1.$$
>
> 1. If $\alpha \in K$,then ψ gives an isomorphism from $E_{ns}(K)$ to $K\times$, considered as a multiplicative group.
> 2. If $\alpha \notin K$,then ψ gives an isomorphism $E_{ns}(K) \simeq \{u + \alpha v \mid u, v \in K, u^2 - av^2 = 1\}$, where the righthand side is a group under multiplication.

For case i., by translating the curve, the triple root is at $(x,y) = (0,0)$ and the equation of the curve is $y^2 = x^3$. By theorem 2.30, which defines the mapping $\phi(P(x,y)) = x/y$, the group is then isomorphic to an additive group, and the ECDLP is reduced to DLP on $(\mathbb{F}_p, +)$. $Q = cP \implies c = \phi(Q)\phi(P)^{-1}$. In this case, the discrete logarithm problem is easier.

For case ii., by translating the curve, the double root is at $(x,y) = (0,0)$ and the equation of the curve is $y^2 = x^2(x+a)$. By theorem 2.31, the mapping has the property $\psi(cP) = \psi(P)^c$, and hence the elliptic curve addition becomes multiplication of elements in $K^\times$ or in a quadratic extension of K. When mapped to a multiplicative group, and the ECDLP is reduced to DLP on $(\mathbb{F}_p, \times)$. There are several methods to solve this problem, like the Baby-step Giant-step algorithm and Pollard's rho algorithm. In this case, the DLP after mapping is rather hard but not harder than the ECDLP.

Therefore, in both cases, the discrete logarithm problem is not harder than of that in a finite field.

3.2. Exercises from the slides

Textbook RSA

REF: https://en.wikipedia.org/wiki/RSA_(cryptosystem)#Proofs_of_correctness

a)

We want to show that $m^{ed} \equiv m \pmod{pq}$ for every integer m when p and q are distinct prime numbers and e and d are positive integers satisfying $ed \equiv 1 \pmod{\phi(pq)}$.

By the Chinese remainder theorem, to check whether two numbers are congurent under pq, it suffices to check whether they are congurent under p and q separately.

Since the Euler totient function $\phi(pq) = (p-1)(q-1)$, we can write $ed - 1 = h(p-1) = k(q-1)$ for some non-negative integers h and k.

To show $m^{ed} \equiv m \pmod{p}$, consider 2 cases:

1. If $m \equiv 0 \pmod{p}$, m is a multiple of p. Thus m^{ed} is a multiple of p. So $m \equiv 0 \equiv m \pmod{p}$.
2. If $m \not\equiv 0 \pmod{p}$,

$$m^{ed} = m^{ed-1}m = m^{h(p-1)}m = (m^{p-1})^k m \equiv (1)^k m \equiv m \pmod{p}$$

, where we used Fermat's little theorem, which states that $a^p - 1 \equiv 1 \pmod{p}$ for any integer a and prime p, to replace $m^{p-1} \bmod p$ with 1.

Similarly, we can show $m^{ed} \equiv m \pmod{q}$.

1. If $m \equiv 0 \pmod{q}$, m^{ed} is a multiple of q. So $m \equiv 0 \equiv m \pmod{q}$.
2. If $m \not\equiv 0 \pmod{q}$,

$$m^{ed} = m^{ed-1}m = m^{k(q-1)}m = (m^{q-1})^k m \equiv (1)^k m \equiv m \pmod{q}$$

QED.

Elliptic Curve Digital Signature Algorithm (ECDSA)

REF: https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Correctness_of_the_algorithm; https://andrea.corbellini.name/2015/05/30/elliptic-curve-cryptography-ecdh-and-ecdsa/; https://bitcoin.stackexchange.com/questions/73622/how-do-you-derive-the-private-key-from-two-signatures-that-share-the-same-k-valu

> Here I have omitted "$\bmod n$" for brevity, and also because the subgroup generated by G has order n, "$\bmod n$" is superfluous.

a)

By definition, $Q = d \times G$. We can write:

$$\begin{aligned} P' &= u_1 \times G + u_2 \times Q \\ &= u_1 \times G + u_2 \times d \times G \\ &= (u_1 + u_2 \times d) \times G \end{aligned}$$

Then, using the definitions of u_1, u_2 and s, we continue to write:

$$\begin{aligned} P' &= (u_1 + u_2 \times d) \times G \\ &= (H(m)s^{-1} + rs^{-1}d) \times G \\ &= s^{-1}(H(m) + rd) \times G \\ &= (k^{-1}(H(m) + rd))^{-1}(H(m) + rd) \times G \\ &= k \times G \end{aligned}$$

This is same equation for $P = k \times G$ in $S_{IGN}(m)$. QED.

b)

If two different messages m_1, m_2 are signed with the same nounce k, the signatures have and the same r as $P = (x, y) = k \times G$ and $r = x \bmod n$. The values of s_1 and s_2 are therefore:

$$\begin{aligned} s_1 &= k^{-1}(H(m_1) + rd) \\ s_2 &= k^{-1}(H(m_2) + rd) \end{aligned}$$

From which we derive:

$$\begin{aligned} s_1 \times k &= H(m_1) + rd \\ s_2 \times k &= H(m_2) + rd \end{aligned}$$

And hence we can find:

$$\begin{aligned} (s_1 - s_2) \times k &= H(m_1) - H(m_2) \\ k &= \frac{H(m_1) - H(m_2)}{s1 - s2} \end{aligned}$$

Finally, calculate the private key:

$$\begin{aligned} d &= r^{-1}(s_1 \times k - H(m_1)) \\ &= r^{-1}(s_1 \times \frac{H(m_1) - H(m_2)}{s1 - s2} - H(m_1)) \\ &= \frac{H(m_1)s_2 - H(m_2)s_1}{r(s_1 - s_2)} \end{aligned}$$

4. Applications of PRG

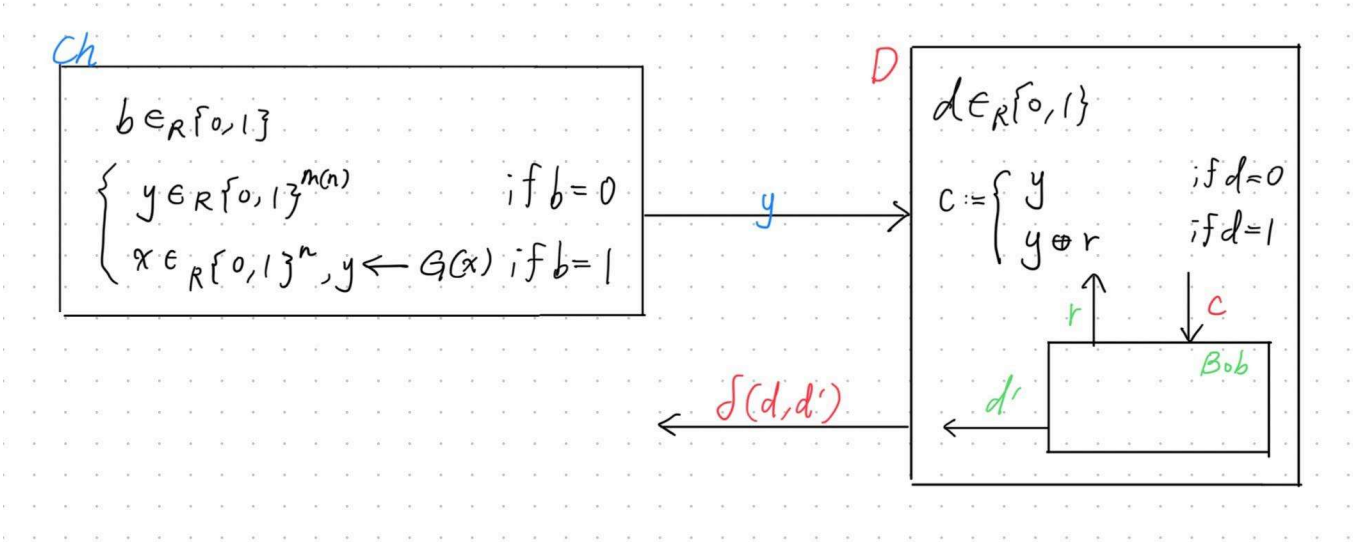
4.1. Locked boxes

REF: *A Graduate Course in Applied Cryptography*

a)

Alice can open the commitment c to both 0 and 1 if and only if she can find x_0 and x_1 such that $c = G(x_0) = G(x_1) \oplus r$, which implies that $G(x_0) \oplus G(x_1) = r$. Let's call this kind of r's "problematic". However, there is at most one problematic r for any pair of seeds (x_0, x_1). The number of pairs of seeds is $(2^n)^2$, and therefore the number of problematic r's is $(2^n)^2$. Since there are 2^{3n} possible values for r, the probability that a randomly chosen r will be the one that allows Alice to cheat is at most $\frac{(2^n)^2}{2^{3n}} = 2^{-n}$.

b)



With Bob, we can construct a D, which redirects y sent from Ch to Bob and use $d \in_r \{0,1\}$ and d' returned by Bob to decide the output:

$$b' = \delta(d, d') = \begin{cases} 0, & \text{if } d \neq d'. \\ 1, & \text{if } d = d'. \end{cases}$$

When $b = 1$, Bob is given $y \leftarrow G(x)$ with $x \in_R \{0,1\}^n$, it is the same as Game 3 and Bob has a winning probability of $1/2 + \mu$; when $b = 0$, Bob is given $y \in_R \{0,1\}^{3n}$, and Bob can only guess randomly, resulting a winning probability of $1/2$. Therefore, as $b \in_R \{0,1\}$, the the winning probability of this D is $\Pr[b' = 1 | b = 1] + \Pr[b' = 0 | b = 0] = (1/2)(1/2 + \mu) + (1/2)(1/2) = 1/2 + \mu/2$.

4.2. An open problem

a)

Consider the decision problem:

$$L = G(\{0,1\}^*) = \{G(x) : x \in \{0,1\}^*\}$$

If $G : \{0,1\}^n \to \{0,1\}^{m(n)}$ is a PRG, then by one the requirements of PRGs:

> G is pseudorandom; that is, for every probabilistic polynomial time (PPT) distinguisher D, D wins Game 1 with probability $\frac{1}{2} + \nu(n)$, where ν is negligible.

Suppose there exists A that returns 0 or 1 in at most $t(|y|)$ steps on input y, t is a polynomial, and $y \in L \iff A(y) = 1$. Then with A, we can construct a D:

$$b' = D(y) = A(y)$$

Let's analyze the probability of winning of this distinguisher. With $m(n) > n$, and as it is possible that $G(x_1) = G(x_2)$ where $x_1 \neq x_2$:

$$\Pr[b' = 1 | b = 1] = \frac{1}{2}$$
$$\Pr[b' = 1 | b = 0] = \frac{\#\text{ of } y \leftarrow G(x)}{\#\text{ of all } y} \leq \frac{2^n}{2^{m(n)}} \leq \frac{2^n}{2^{n+1}} = \frac{1}{2}$$
$$\Pr[b' = 0 | b = 0] = 1 - \frac{2^n}{2^{m(n)}} \geq 1 - \frac{1}{2} = \frac{1}{2}$$

D wins Game 1 with probability: $\Pr[\text{D wins}] = \Pr[b' = 1 | b = 1] + \Pr[b' = 0 | b = 0]$, since if $b = 1$, i.e. $x \in_R \{0,1\}^n,\ y \leftarrow G(x)$, D will know that $y \in L$ with the help from A, and if $b = 0$, D would guess 0 and will lose only if y collides with some $G(x)$. As $\Pr[\text{D wins}] \geq \frac{3}{4} > \frac{1}{2} + \nu(n)$, there is a contradiction, and hence we know that there does not exist an algorithm A that solve L in polynomial time. Therefore, $L \notin \mathbf{P}$.

Then, since there exists a polynomial-time algorithm:

$$V(x, y) = \begin{cases} 1, & \text{if } G(x) = y \\ 0, & \text{if } G(x) \neq y \end{cases},$$

and we know that

$$y \in L \iff \exists x \text{ s.t. } V(y, x) = 1,$$

where $y \in \{0,1\}^{m(n)}$, we can say $L \in \mathbf{NP}$.

With $L \notin \mathbf{P}$ and $L \in \mathbf{NP}$, we conclude that if $G : \{0,1\}^n \to \{0,1\}^{m(n)}$ is a PRG, then $\mathbf{P} \neq \mathbf{NP}$.

5. Simple Crypto

a)

REF: https://en.wikipedia.org/wiki/Affine_cipher
FLAG: `CNS{51MP13_M47H_70_8r34K_4FF1N3_a3e4aa86dc09bf988d0596c4e83a6af2}`
For Affine cipher, the encryption function is

$$E(x) = ax + b \mod m,$$

where $m = 256$, the size of the ascii alphabet, and a and m are coprime. Since the first two letters in the plaintext is provided in the hint, we can find the a, b used for encryption and decrypt the cipher text with function

$$D(x) = a^{-1}(x - b) \mod m,$$

with $m = 256$ and a^{-1} being the modular multiplicative inverse of $a \mod m$.

b)

FLAG: `CNS{8rU73_F0rC3_4r3_P0551813_70_8r34K_Z16Z46_50b012399a6eb9004fa2ae62d276cec2}`
I conver the hex-encoded ciphertext to ascii text, and use this tool: https://cryptii.com/pipes/rail-fence-cipher to decrypt the passphrase by trial & error.

c)

REF: https://en.wikipedia.org/wiki/Polybius_square
FLAG: `CNS{P01Y81U5_0N1Y_H4V3_5_UN1QU3_CH4r4C73r5_1N_C1PH3r73X7_ca59a45ed946684762b87dd9ef90eeba}`
The passphrase can be solved by replacing the five letters from the ciphertext (converted from hex to ascii) with permutations of numbers from 1 to 5, and use every two numbers as the coordinate on the polibius square. Output the decoded result of all the permutations to a file `eve.txt`, and find the only passphrase that contains meaningful words with human-labor.

d)

FLAG: `CNS{07P_4CH13V35_P3rF3C7_53CUr17Y_UND3r_455UMP710N5_b7dd544750014b538024ebe801788feb}`
First, *decode* the passphrase with base64. As for *decrypting* the secret, since I already know 4 characters of the complete plaintext: `CNS{`, I can obtain 4 characters of the key with XOR. I guess the other 2 characters of the key from the 256 ascii codes, and output every **printable** decrypted plaintext to a file `admin.txt`. Finally, with human-labor, I find the flag from the line of plaintext containing meaningful words only.

6. Simple RSA

The tool I used for this problem: https://github.com/RsaCtfTool/RsaCtfTool

a)

FLAG: `CNS{345Y_F4C70r1Z4710N}`
Since the n is small in this challenge, I used the factordb attack along with the tool to solve for the flag:
`./RsaCtfTool.py --attack factordb`

b)

REF: https://crypto.stackexchange.com/questions/1614/rsa-cracking-the-same-message-is-sent-to-two-different-people-problem/1616#1616
FLAG: `CNS{7W0_C1PH3r73X7_W111_8r34K_r54?!_79f17b6d9c07fb154c6dcdb92ab047a7}`
There are 2 ciphertexts encrypted with the same n and 2 different e's, therefore we can find a, b s.t. $a \cdot e_1 + b \cdot e_2 = 1$, and decrypt with $m \equiv c_1^a \cdot c_2^b \mod n$. Luckily this attack is provided by the tool as well:
`./RsaCtfTool.py --attack same_n_huge_e`

c)

REF: https://docs.xanhacks.xyz/crypto/rsa/08-hastad-broadcast-attack/;
https://crypto.stackexchange.com/questions/6713/low-public-exponent-attack-for-rsa;
https://en.wikipedia.org/wiki/Chinese_remainder_theorem
FLAG: `CNS{CH1N353_r3M41ND3r_r3P347_8r04DC457_4774CK_edc38122a288aa9922617b5a6a08ac61}`
The Hastad's broadcast attack works when the e is small and we can obtain e-pairs of n and ciphertext. For the following systen of equations:

$$m^e \equiv c_i \mod n_i, \text{ where } i \in [1, 7]$$

I use the Chinese Remainder Theorem to solve $x = m^e \pmod{\prod_{i=1}^{7} n_i}$, and obtain $m = \sqrt[e]{x}$.

d)

REF: https://crypto.stackexchange.com/questions/68641/rsa-encryption-extremely-large-public-exponent; https://docs.xanhacks.xyz/crypto/rsa/07-wiener-attack/
FLAG: `CNS{816_e_4ND_W13N3r_4774CK_1N_7H3_NU75H311_0f2146bda49df4206f344cd29b0503fa}`
As the e is huge and $e \cdot d = 1 \mod n$, the private key d must be small in this case, and hence I can obtain the plaintext with wiener attack:
`./RsaCtfTool.py --attack wiener`

7. POA

a)

FLAG: `CNS{p4dd1n9_0r4cl3_4tt4ck_15_3v1l}`
Implement an oracle padding attack by guessing the plaintext byte by byte and checking correctness with the oracle from the server (`"Invalid message/Message sent"`).

b)

FLAG: `CNS{f0r61n6_r3qu3575_4r3_5up3r_345y}`
The message needed to obtain the flag is `"Request nounce: so_random_nounce; Sender: TA; Message: Please send over the 2nd flag"` ; to achieve this, I put `"TA; Message: Please send over the 2nd flag"` in the "Your Name: " field and append a fake padding in the same style of the server, which makes the server omit the content after this fake padding when it unpads the message.

8. CNS Store

First of all, break into the store with a sha-256 value partial-colliding with the string of the length of 6, provided by the server. I append a integer counter, which increases by 1 every time, after the string `"CNS2024"` , and find the collision by brute-force.

a)

FLAG: `CNS{S1mp!e_H45h_C011!510n_@7tacK!}`
There are three things you can buy at this store:

1. Buying a product, of which the name must contain the string `"CNS2024"` , and the sha-1 value of it would be store on the `shelf` . The maximum amount of a product is 10, and the maximum amount you can purchase once is also 10. There exist another list `sold` , so you cannot buy the product in different purchases. Therefore, if you find a pair of sha-256 collision, you can buy the same product 20 times, costing you 5*10*2 = 100 dollars and resulting a negative amount (-10) of product on the `shelf` .
2. Buying the lottery cost you 5 times the amount of products on the shelf. After the purchases of the colliding products, buying the lottery 3 times will cost you 3*5*(-10) = -150 dollars
3. Buying the flag costs 150 dollars. With the $100 I initially received and the 2 purchases of products and 3 purchases of lottery, I have 100 - 100 - (-150) = 150 dollars now!

The pair of sha-1 collision I used is from https://shattered.io/, and I put the string `"CNS2024"` in front of it.

b)

FLAG: `CNS{H@ppy_B!r7hDaY_t0_Y0U!}`

This is basically the same as the last subproblem with sha-256 instead of sha-1. Hence I find a pair of sha-256 collision with the similar method for breaking into this store (the only difference is there is a `key` provided by the server that must be also included in the plaintext, i.e. product name), and then carry out the same sequence of purchases as the previous part to obtain the flag.

c)

FLAG: `CNS{https://www.youtube.com/watch?v=YFJowRNfMGI}`

With known range of message and key length, I can use the hash length extension attack to modify the ID. The tool I used is https://github.com/viensea1106/hash-length-extension, which generates a pair of ID and signature to fake the `Identity=admin` part of message.