

汇总报告

人员分工表：

姓名：郝梅

学号：202100150027

贡献：个人组队，所有项目独立完成

代码仓库地址：

[MayHao33/homework_group_33: Projects of Innovation and entrepreneurship practice course \(github.com\)](https://github.com/MayHao33/homework_group_33)

项目完成情况：

- ✓ Project1: implement the naïve birthday attack of reduced SM3
- ✓ Project2: implement the Rho method of reduced SM3
- ✓ Project3: implement length extension attack for SM3, SHA256, etc.
- ✓ Project4: do your best to optimize SM3 implementation (software)
- ✓ Project5: Impl Merkle Tree following RFC6962
- ✗ Project6: impl this protocol with actual network communication
- ✓ Project7: Try to Implement this scheme（仅部分）
- ✓ Project8: AES impl with ARM instruction（仅部分）
- ✓ Project9: AES / SM4 software implementation
- ✓ Project10: report on the application of this deduce technique in Ethereum with ECDSA（仅部分）
- ✓ Project11: impl sm2 with RFC6979
- ✗ Project12: verify the above pitfalls with proof-of-concept code
- ✓ Project13: Implement the above ECMH scheme
- ✓ Project14: Implement a PGP scheme with SM2
- ✗ Project15: implement sm2 2P sign with real network communication
- ✓ Project16: implement sm2 2P decrypt with real network communication（有错误）

√Project17: 比较 Firefox 和谷歌的记住密码插件的实现区别

×Project18: send a tx on Bitcoin testnet, and parse the tx data down to every bit,
better write script yourself

×Project19: forge a signature to pretend that you are Satoshi

×Project21: Schnorr Bacth

√Project22: research report on MPT

项目情况概述:

Project 1: implement the naïve birthday attack of reduced SM3	
运行环境	硬件环境: 处理器: 11th Gen Intel(R) Core(TM) i5-1135G7@2.40GHz 内存: 16.0GB(10.8GB 可用) 软件环境: 操作系统: win11 编译器: Python 3.10
运行时间	找到前 8bit 相同, 用时 0.04833245277404785s 找到前 16bit 相同, 用时 0.9354827404022217s 找到前 32bit 相同, 用时 95.54628419876099s
实现方式	构建两个随机数表, 对两个随机数表中的数值求其哈希值, 构建两个哈希表。 在构建哈希表的过程中, 不断对两个哈希表求交集 (即寻找碰撞), 这样能在找到碰撞的第一时间结束攻击。
实现效果	通过建立列表寻找碰撞, 以空间换取时间, 提高了攻击速度。

Project 2: implement the Rho method of reduced SM3	
运行环境	硬件环境: 处理器: 11th Gen Intel(R) Core(TM) i5-1135G7@2.40GHz 内存: 16.0GB(10.8GB 可用) 软件环境: 操作系统: win11 编译器: Python 3.10
运行时间	找到前 8bit 相同, 用时 0.76735s

实 现 效 果	添加的额外信息并无任何要求，成功地实现了对于 SHA-256 与 SM3 的长度扩展攻击。
------------------	---

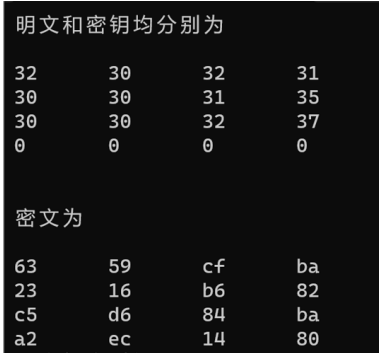
Project 4: do your best to optimize SM3 implementation (software)	
运行环境	硬件环境： 处理器：11th Gen Intel(R) Core(TM) i5-1135G7@2.40GHz 内存：16.0GB (10.8GB 可用) 软件环境： 操作系统：win11 编译器：Visual Studio 2022
运行时间	未优化之前，用时 0.0243ms 优化后，用时 0.0142ms
实现方式	1. 使用位运算：代码中使用位运算来代替乘法、除法和模运算，因为位运算在大多数处理器上执行得更快。 2. 数组预计算：代码中使用预计算的常量数组 T 来避免在每次循环中重新计算常量，减少了重复计算的开销。 3. 减少内存分配：代码中避免了不必要的内存分配操作。 4. 循环展开：代码中对循环进行了展开优化，减少了循环迭代的次数，减少了分支预测和循环控制的开销。
实现效果	相比最初 SM3 的执行时间，优化后的执行时间缩短。

Project 5: Impl Merkle Tree following RFC6962	
运行环境	<p>硬件环境：</p> <p>处理器：11th Gen Intel (R) Core(TM) i5-1135G7@2.40GHz</p> <p>内存：16.0GB(10.8GB 可用)</p> <p>软件环境：</p> <p>操作系统：win11</p> <p>编译器：Python 3.10</p>
运行结果	<pre>Root Hash: 4ab0728209309b6f916ccf4d30b65abec0fe5a1ebdf2e9e049322f29056b7477 Proof for leaf 2 : ['6669667468', '062316eee7370f9459714c52a4634782ca9f08dbef850ce76366bfd3dd5dd782']</pre>
实现方式	<ul style="list-style-type: none"> ■ 代码导入 Python 的 hashlib 库，用于计算 SHA-256 哈希值 ■ <code>init(self, leaves)</code>: 类的初始化方法。接收一个叶子节点的列表，并将其存储在 <code>self.leaves</code> 中。 ■ <code>build_tree(self)</code>: 构建默克尔树。使用 <code>self.levels</code> 列表来存储每个层级的节点列表。首先将叶子节点列表加入 <code>self.levels</code>，然后在一个 <code>while</code> 循环中逐层构建默克尔树，直至到达默克尔根。 ■ <code>build_level(self, level)</code>: 构建树的一层。函数接 <code>leve</code> 作为输入，表示当前层级的节点列表。通过对当前层级进行迭代，每次处理两个节点（如果有两个），并将它们的组合结果添加到下一层级的节点列表 <code>next_level</code> 中。函数返回下一层级的节点列表。 ■ <code>hash_children(self, left_child, right_child)</code>: 对子节点求哈希。将左子节点和右子节点组合在一起，然后使用 SHA-256 哈希算法对组合结果进行计算，并返回哈希值。 ■ <code>get_root(self)</code>: 获取默克尔树的根节点。如果 <code>self.levels</code> 为空，则返回 <code>None</code>；否则，返回 <code>self.levels</code> 中最后一层的第一个节点，即根节点。 ■ <code>get_proof(self, leaf_index)</code>: 获取指定叶子节点的验证证明，即给出到达叶子节点的路径。接收一个叶子节点的索引作为参数。方法会遍历每个层级的节点列表，根据叶子节点的索引来生成验证证明。如果叶子节点的索引是偶数，则从同一层级的下一个节点添加到证明中，并将叶子节点的索引除以 2。如果叶子节点的索引是奇数，则从同一层级的上一个节点添加到证明中，并将叶子节点的索引减 1 再除以 2。函数返回生成的证明列表。
实现	<p>代码实现了通过给出叶子节点的赋值，使用 <code>build_tree</code> 方法构建默克尔树，打印输出默克尔根。</p> <p>同时还实现了获取指定叶子节点的验证证明的功能，即获取达到叶子节点路径上的哈希值并打印输出。</p>

效果	
----	--

Project 7: Try to Implement this scheme (仅实现部分)	
运行环境	<p>硬件环境：</p> <p>处理器：11th Gen Intel(R) Core(TM) i5-1135G7@2.40GHz</p> <p>内存：16.0GB(10.8GB 可用)</p> <p>软件环境：</p> <p>操作系统：win11</p> <p>编译器：Python 3.10</p>
运行结果	<p>Credential Chain:</p> <p>09df68f6d02f3cc799f94f27a507427e3015cbfa47655ca36c1330a8b60b72fb c539ddcccb8ee527f7adf8d019f8b9cb0ee156d8790b5d61b9c6622c069a4793 d1dd1b40c44aca6627425b03f43fc1e8d6de2be03cbe69b733e9742c078f2bc5 7a7bd50855e0fa82f6aea1e47addf919f533bf72bc6a9e4a694dc114c59af468 685965410dba2f37c40ce47f2d79f6590bf71727c375b2e6815ed40fd4ccbeed 7ee4b63e13bdb0827fc653f22489df95bf2dbc9f321119088f2c5d925594b683 bd706678cd61c28e00e51bce44e64b6788e94521fadc3e0f6517752531e5e0d4 b3526e4b5cf4fa150fe86e09f0750f8c7007b474c8db52f48cfef6cb8acfe7fd a614d4946d016ac26663c5a0d4a451e8211da4d8aeb408505e8a71b310ba140d 0c50271b18f5c7cfff616fb89d28b3e27e7be070f8371f1f2bce4032a76525c1</p> <p>Range proof is invalid!</p>
实现方式	<ul style="list-style-type: none"> 函数 <code>sha256(input)</code>：对输入字符串执行 SHA-256 哈希，并返回哈希值的十六进制表示。 函数 <code>generate_credential_chain(initial_value, num_credentials)</code>：生成 HashWires 链。从初始值开始，使用 SHA-256 哈希生成一系列凭证，并将它们存储在列表中。函数返回凭证链。 函数 <code>verify_range_proof(credential_chain, proof_value, min_value, max_value)</code>：进行范围证明。给定一个凭证链、证明值和最小/最大值，函数遍历凭证链，检查证明值是否在指定范围内。如果找到匹配的证明值，则检查计数器是否在指定范围内。如果范围证明有效，函数返回 True，否则返回 False。
实现效果	<p>仅实现了基于 HashWires 的范围证明，其中 <code>generate_credential_chain</code> 函数可生成凭证链。</p>

Project 8: AES impl with ARM instruction（仅部分）	
运行环境	电脑无法运行 ARM 指令集，因此并未运行。
实现方式	<ul style="list-style-type: none">■ 函数 <code>aes_encrypt</code>: AES 加密函数。它接受一个 128 位的输入状态 (<code>state</code>) 和一个 128 位的密钥 (<code>key</code>)，并使用 AES 算法对输入状态进行加密操作。函数首先将输入状态和密钥加载到寄存器中，然后执行多轮的 AES 操作，最后将加密后的状态存储回输入状态。■ 数据段: 代码中定义了一些数据，包括密钥、输入状态和轮密钥。密钥和输入状态都是由 16 个字节组成的数组，轮密钥是一个 16*11 个字节的数组，用于存储每一轮的轮密钥。
实现效果	由于自身对 ARM 指令集以及 AES 指令扩展了解较少，仅给出了 AES 加密的一部分实现，还缺少了密钥扩展、字节代替等必要的步骤。由于并未运行，代码也可能有错误存在。

Project 9: AES / SM4 software implementation	
运行环境	<p>硬件环境：</p> <p>处理器：11th Gen Intel(R) Core(TM) i5-1135G7@2.40GHz</p> <p>内存：16.0GB(10.8GB 可用)</p> <p>软件环境：</p> <p>操作系统：win11</p> <p>编译器：Python 3.10、Visual Studio 2022</p>
运行结果	<p>AES</p>  <p>SM4</p> <p>明文：000000000000000000000000202100150027</p> <p>密文：e0c522c4e0c522c4e7faf0314a974f36</p>
运行时间	<p>AES 一次加密：0.419236ms</p> <p>SM4 一次加密：0.200500ms</p>
实现方式	<ul style="list-style-type: none">■ AES 加密实现已在课程密码学引论中学习，在此不再赘述。■ SM4 加密实现代码解释<ul style="list-style-type: none">● 函数 left_shift：循环左移操作。● 函数 sm4_key_schedule：SM4 的密钥扩展算法。它接受一个 128bit 密钥，将其拆分为 4 个 32bit 的子密钥，利用循环左移和异或操作生成 32 个轮密钥 (rk)。● 函数 sm4_round：SM4 的一轮加密操作。它接受一个 32 位的输入 (x) 和一个轮密钥 (rk)，对输入进行一轮加密操作，返回加密后的结果。● 函数 sm4_encrypt：SM4 的加密函数。它接受一个输入数据和一个密钥，将输入数据划分为 32 位的块，使用轮密钥对每个块进行多轮的加密操作，并将加密后的结果拼接成输出数据。

实 现 效 果	实现了 AES 与 SM4 加密算法，输入明文与密钥，可有效地加密得到密文。
------------------	--

Project 10: report on the application of this deduce technique in Ethereum with ECDSA (仅部分)	
运 行 环 境	硬件环境： 处理器：11th Gen Intel (R) Core(TM) i5-1135G7@2.40GHz 内存：16.0GB(10.8GB 可用) 软件环境： 操作系统：win11 编译器：Python 3.10
运 行 结 果	<pre> Message: Hello,May! Signature: e39aa7af941b2601cbe571dea16bb74a3f85e47c96ddc81dc7cf3450eededf31a64f2 2dfe3185318add24e765f52828d3816b70d060bff6e2a715e104664f596 Is Valid: True </pre>
实 现 方 式	<ol style="list-style-type: none"> 1. 生成私钥：通过调用 <code>ecdsa</code> 库中的 <code>SigningKey.generate()</code> 方法，使用 SECP256k1 曲线生成一个私钥对象 <code>private_key</code>。SECP256k1 是一种椭圆曲线，广泛用于加密货币（比特币）的签名算法。 2. 获取公钥：通过调用私钥的 <code>get_verifying_key()</code> 方法，获取与私钥对应的公钥对象 <code>public_key</code>。公钥是为了验证签名。 3. 生成消息：定义了一个字符串消息 <code>message</code>，作为签名和验证的原始消息，该消息可以由用户输入。 4. 对消息进行签名：使用私钥对象的 <code>sign()</code> 方法对消息进行签名。<code>sign()</code> 方法的接收参数为字节形式，因此先使用 <code>encode()</code> 方法将消息字符串转换为字节。 5. 验证签名：通过调用公钥对象的 <code>verify()</code> 方法，使用签名和消息的字节表示进行验证。<code>verify()</code> 方法返回一个布尔值，表示签名是否有效。 6. 打印输出：通过 <code>print()</code> 语句将消息、签名和验证结果输出。

实 现 效 果	了解了 ECDSA 算法的具体描述与算法原理，对推导技术有了初步的认识。这里借助 Python 的 ecdsa 库仅实现了椭圆曲线数字签名算法（ECDSA）的生成和验证。
------------------	---

Project 11: impl sm2 with RFC6979	
运 行 环 境	硬件环境： 处理器：11th Gen Intel (R) Core(TM) i5-1135G7@2.40GHz 内存：16.0GB (10.8GB 可用) 软件环境： 操作系统：win11 编译器：Python 3.10
运 行 结 果	<pre> Message: Hello, May! Signature: b'iv\xa1 \&\x99B\xe3\x10\xc4\xa3\xee\xb0?\x90\x0bS{\x02\xf30\xbd\xe2\xd2\xb7-\xfa:\`rq\xf8\xa1\xa8\x87\xb8\xb6)\xf8(\xd1\x84"*\xa96\xcb\x136\xc3u\xff\x89TKA.\xa8\xed\xb28\xc8\x94c' The signature is valid: True Tamper signature,then: The signature is valid: False </pre>
实 现 方 式	<ul style="list-style-type: none"> ■ 使用 hashlib 模块中的 sha256 函数，ecdsa 模块中的 SigningKey 函数和 SECP256k1 曲线。SECP256k1 是一种椭圆曲线，广泛用于加密货币（比特币）的签名算法。 ■ 函数 generate_sm2_keypair：生成 SM2 密钥对，使用 SECP256k1 曲线生成一个随机私钥，然后通过私钥计算对应的公钥。 ■ 函数 sm2_sign：使用私钥对消息进行签名。计算消息的哈希值，然后使用 RFC6979 确定性签名生成算法，使用私钥对哈希值进行签名，并返回签名结果。 ■ 函数 sm2_verify：使用公钥对消息和签名进行验证。计算消息的哈希值，然后使用公钥对签名进行验证，如果验证成功返回 True，否则返回 False。
实 现 效 果	实现了 SM2 的数字签名功能和验证功能。签名消息可由用户输入，得到对应的数字签名后，并根据公钥对签名进行验证。若篡改签名，则无法通过签名验证。

Project 13: Implement the above ECMH scheme	
运行环境	硬件环境： 处理器：11th Gen Intel(R) Core(TM) i5-1135G7@2.40GHz 内存：16.0GB(10.8GB 可用) 软件环境： 操作系统：win11 编译器：Python 3.10
运行结果	===== RESTART: D:\May_Hao\MAY.SEMESTER\创新创业实践\homework\project 13\ECMH.py =====
实现方式	ECMH('Hello,May!') = (17538674033683862238579825180956273276076570998349360457036461104345977985539, 64487023625635920337360403067548719616968140431420868373896347539298198302677)
实现效果	①定义椭圆曲线的参数 a、b、p、n 和 G。其中，a 和 b 是椭圆曲线的参数，p 是素数，n 是曲线的阶，G 是曲线上的生成点。 ②定义点加法 (point_addition) 和点的乘法 (scalar_multiplication) 的函数。点加法根据椭圆曲线上的加法规则，将两个点 p1 和 p2 相加，得到一个新的点。点的乘法使用二进制表示私钥 k，并通过二进制位的运算来进行点的倍乘操作。 ③定义了 ECMH 函数。该函数首先对输入的消息进行 SHA-256 哈希计算，得到一个哈希值。然后，使用该哈希值作为私钥，通过点的倍乘运算，计算出对应的公钥点。最后，返回公钥点作为 ECMH 的输出。 ④定义消息变量，调用 ECMH 函数计算出相应的 ECMH 结果，并输出。
	实现了 ECMH 结构，消息 message 由用户输入，能调用 ECMH 函数计算出相应结果。

Project 14: Implement a PGP scheme with SM2	
运行环境	<p>硬件环境：</p> <p>处理器：11th Gen Intel (R) Core(TM) i5-1135G7@2.40GHz</p> <p>内存：16.0GB (10.8GB 可用)</p> <p>软件环境：</p> <p>操作系统：win11</p> <p>编译器：Python 3.10</p>
运行结果	<pre>===== RESTART: D:\Python\PGP_SM2.py ===== encrypt_value: b'})z\x9e\xae~\x88\xee\x05\xda\xd3\xa8\xc2\xe2\x85\x0ea' enc_data: b'\xef\x97-\xb4\x9f\x83F\xc5\xb6\x14\x15K\xd6fw\xe3\xf4\xe6\xc8\xf8\x0c\xe7E\x89M\xd4\x87?\xa6\x8b\x845\xb0\xdc\xddM\xf4\xf8\xd4\xb3\xcb,\xe4\x11 \xf9\xd65\x00\x9e\xe7q\x06\x91\xa2j\xd6\x02\xee<\x92\x00\x90\xa3\xd9V\x7f\xcf\xe3\x08\xeb\xfe\xc78\x81\xd2\xbc\t\x13I\x1e\xc5\xd1\xebM\xfc\xe9f\xa4l\xd5\x1e\x9c\xdb\xaard=\x1b\xfd,f\xd0~\t_p\xe3B0yT' decrypt_value: b'' k: b'}\xb7 \x94U\x8f\xe3a\x9fru\xdc\x7f@\xfb9'</pre>
运行时间	整个执行 PGP 流程用时 0.0510842s
实现方式	<ul style="list-style-type: none"> ■ 代码借助 Python 中 gmssl 库，提供了对 SM2 算法的支持，用于生成密钥对、加密与解密 ■ 变量阐释 <ul style="list-style-type: none"> ● private_key 和 public_key: SM2 密钥对的私钥和公钥，以十六进制字符串表示。 ● iv: SM4 算法的初始向量，用于对称加密过程。 ● crypt_sm4: SM4 对称加密算法的实例，用于进行对称加密和解密。 ● sm2_crypt: SM2 非对称加密算法的实例，用于进行非对称加密和解密。 ■ PGP_Enc 方法: encrypt 方法接受消息和密钥作为输入，并返回加密后的消息和使用 SM2 加密的密钥。调用 sm2_crypt.encrypt(k) 使用公钥对密钥进行加密，得到 enc_data。接着，使用 crypt_sm4 设置对称加密算法的密钥，并使用 CBC 模式对消息进行加密，得到 encrypt_value。 ■ PGP_Dec 方法: decrypt 方法接受加密的消息和使用 SM2 加密的密钥作为输入，并返回解密后的消息和解密出来的密钥。使用 sm2_crypt 对使用 SM2 加密的密钥进行解密，得到原始的密钥 k。接着，使用 crypt_sm4 设置对称加密算法的密钥，并对加密后的消息进行解密，得到原始消息 decrypt_value。

实 现 效 果	基于 PGP 的数据加密和解密功能，结合了非对称加密（SM2）和对称加密（SM4）两种算法，可以为用户提供更好的安全性和性能。
------------------	---

Project 16: implement sm2 2P decrypt with real network communication（有错误）	
运 行 环 境	<p>硬件环境：</p> <p>处理器：11th Gen Intel(R) Core(TM) i5-1135G7@2.40GHz</p> <p>内存：16.0GB(10.8GB 可用)</p> <p>软件环境：</p> <p>操作系统：win11</p> <p>编译器：Python 3.10</p>
运 行 结 果	<pre>PS D:\Python> python main.py Waiting for sender to connect... Connected to: 127.0.0.1:9664 Traceback (most recent call last): File "D:\Python\main.py", line 56, in <module> main() File "D:\Python\main.py", line 51, in main plaintext = decrypt_ciphertext(ciphertext, private_key) File "D:\Python\main.py", line 35, in decrypt_ciphertext plaintext = private_key.decrypt(</pre>
实 现 方 式	<ul style="list-style-type: none"> ■ 接收方 ● 定义 load_private_key 函数，用于从 PEM 文件中加载私钥。 ● 定义 establish_connection 函数，用于建立网络连接。该函数创建一个 TCP 监听套接字，等待发送方连接。 ● 定义 receive_ciphertext 函数，用于接收发送方发送的密文。 ● 定义 decrypt_ciphertext 函数，用于解密密文。函数使用私钥对象的 decrypt 方法进行 SM2 解密，并返回解密后的明文。 ● 定义 close_connection 函数，用于关闭网络连接。 ■ 发送方 ● 使用 socket 库的 socket 函数创建一个套接字对象。 ● 使用套接字对象的 connect 方法连接到接收者的主机和端口号。 ● 使用套接字对象的 send 方法发送密文数据。 ● 使用套接字对象的 close 方法关闭连接

实现效果	<p><code>generate_key.py</code> 完成了密钥的生成，并将其存储在 <code>private_key.pem</code> 文件中。</p> <p><code>main.py</code> 所代表的接收端无法正确解密，只能正确执行等待连接与连接成功，后续由于解密函数的错误无法确定其运行结果。</p> <p><code>sender.py</code> 可完成建立连接，发送数据，但由于接收者处解密受阻，故无法继续执行。</p>
------	---

Project 17: 比较 Firefox 和谷歌的记住密码插件的实现区别	
调查结果	<p>通过查询资料，了解了 Firefox 和谷歌的记住密码插件的实现区别。</p> <p>方便根据个人需求，选择更合适的浏览器：</p> <ul style="list-style-type: none">■ 若注重密码管理与隐私保护，更倾向于选择 Firefox 浏览器■ 若注重跨设备同步与账户集成，更可能选择谷歌浏览器 <p>更进一步理解密码存储与安全机制，帮助评估密码管理的风险，并采取相应的安全措施。</p>

Project 22: research report on MPT	
调查结果	<p>通过查询资料，对默克尔树（Merkle Tree）与前缀树（Trie）的基本概念、原理与实现有了更加深入的了解。在此基础上，对 MPT 的三种节点类型、构建原理有了认识。</p>