Cryptomato: An Interactive Platform for Cryptography Education

Yechan Bae
903515918
Georgia Institute of Technology
yechan@gatech.edu

Yonghwi Jin
903508901
Georgia Institute of Technology
yonghwi@gatech.edu

Seulbae Kim
902780909
Georgia Institute of Technology
seulbae@gatech.edu

Jungwon Lim
903524908
Georgia Institute of Technology
jungwon@gatech.edu

ABSTRACT

Modern computer systems and web services depend greatly on cryptographic schemes for ensuring security and privacy. Even though the use of strong cryptographic schemes appears to be a universal trend, it turns out that many services in practice use a weak scheme, or incorrect implementation and fail to provide adequate security. Given that, cryptanalysis is one of the essential skills for security professionals to cultivate. In this project, we propose Cryptomato, which is an interactive platform that grants students a unique opportunity to learn and exercise fundamental cryptanalysis techniques through hands-on experiences of designing and implementing attacks. Cryptomato complements the existing manual assignments of CS6260 class not only through providing the grade and feedback within seconds but also with its managing functionality and extensibility.

KEYWORDS

Cryptography education; interactive learning

1 INTRODUCTION

Cryptography has come a long way, from classic cipher to the RSA encryption, and it seems apparent that more and more people are becoming aware of the importance of cryptography. For instance, nowadays, TLS encryption over HTTP is the de facto standard for most of the leading websites and web services, including those of the US government [2], and Chase Bank advertises its encryption scheme as a way of securing their customers' privacy [7]. This trend gives the impression that a safe and secure ecosystem is built on top of well-designed cryptographic schemes that users can trust.

Unfortunately, even with the increased awareness and widespread use of cryptography, threats targeting bad cryptographic schemes persist, and successful attacks and associated breaches are being reported frequently [8, 9], leaving thousands of millions of users' privacy at stake.

Given this situation, training and securing cybersecurity professionals who are capable of conducting proper cryptanalysis is important. Georgia Tech's CS6260 has been adequately serving this purpose. During the course, students learn basic to advanced topics on cryptography and exercise the understanding of those concepts through the assignments. The assignments involve paper-based manual work about analyzing the security of the given situation, designing attacks, and evaluating the advantages of the attacks. However, we could identify several limitations that are innate to such paper-based assignments. First, the discrepancy between theory and practice is not handled, meaning that students do not have the opportunity to take one step further and implement the attack they designed to apply in practice. In addition, as the grading and feedback is purely based on the manual efforts of the instructors, students inevitably get delayed, as well as potentially erroneous feedback, not to mention the doubled workload from the instructor side.

One way to complement the existing assignments would be voluntarily participating in various Capture The Flag (CTF) contests or war-games, *e.g.*, DEF CON CTF [3], and exploring the crypto-related challenges. Unfortunately, most of these challenges put too much emphasis on writing a one-time exploit to subvert the system, without requiring any analysis of the efficiency and the advantage of the attacker. Moreover, not all topics covered during the class have corresponding challenges, leaving out a great portion of the essential skills accumulated through the course. We have reached the conclusion that we are in dire need of a cryptography platform, which is tailored for education.

As a remedy, we propose Cryptomato, an interactive platform for cryptography education that can bridge the gap between the theory and practice, and further foster effective learning. Cryptomato can complement or replace the existing assignments by hosting various cryptography challenges, *i.e.*, homework, and letting students design, code, and

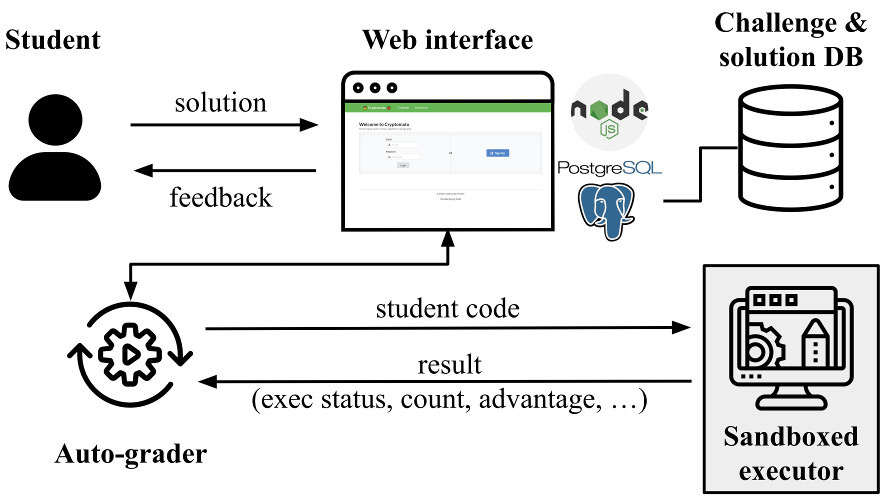


Figure 1: Overview of CRYPTOMATO's architecture and workflow.

submit the adversarial attacks against these challenges. The biggest difference compared to previous approaches is that CRYPTOMATO accepts the attacker code from the student instead of a one-time answer, *i.e.*, a flag. The auto-grader included in CRYPTOMATO immediately evaluates the submitted attack code, performs statistical efficiency analysis, and provides the grade and feedback so that the student can have an opportunity to continue working on the assignment and eventually complete the learning process of the subject matter in one sitting. Multiple benefits are expected here: students can gain hands-on experience regarding the implementation of his or her attacks, which can be seamlessly applied to real-world problems. Instructors also benefit from using CRYPTOMATO, as tweaking or creating new homework problems is made easy, and they can also offload the grading and managing burdens to the auto-grader.

In this report, we describe the design of CRYPTOMATO along with the unique challenges posed during the development of CRYPTOMATO and evaluate the impact of the platform through the ported CS6260 assignments.

2 CRYPTOMATO: DESIGN AND FEATURES

In this section, we focus on the design and features of the novel interactive platform we propose, CRYPTOMATO to foster active learning of cryptography.

2.1 Overview

Figure 1 shows the overview and workflow of CRYPTOMATO. CRYPTOMATO is composed of three main components: (1) front-end web interface, (2) back-end worker nodes including the auto-grader and the sandboxed executor, and (3) challenge & solution database. The web interface and worker nodes reside in separate docker containers, and communicate using gRPC [1].

2.2 Front-end

Web interface. All user interactions take place in the front-end web interface, which is accessible through a web browser. CRYPTOMATO is a standard web service, to which users can register, sign in, browse challenges and associated instructions, submit his or her solution, and get a feedback. Each challenge page elaborates on the problem statement, requirements, and the available schemes, as shown in Figure 3.

Challenge management. CRYPTOMATO front-end facilitates the management of the challenge and solution database. For each challenge, instructors only need to write a manifest (`[chal].toml`) and a test (`[chal].test.py`) for the problem. The manifest file includes the name and the description of the challenge, and the test file contains the test code for the problem. When the front-end starts, it automatically registers all unregistered challenges in the challenge directory and updates the challenge database.

Writing adversary code. In order to solve a challenge in CRYPTOMATO, students need to submit their adversary code written in Python programming language through the web interface. An adversary function takes parameters from the test code, e.g., cipher parameters such as a message length and encryption/decryption oracles. Then, this function should return its "guess", *e.g.*, a guess b or a forged message and tag pair, depending on the problem setting. Our approach allows general construction and evaluation of the adversarial attacker model. Combined with efficiency analysis, it can model various experiments of provable security such as IND-CPA, IND-CCA, and UF-CMA.

Once a user submits the code through the web interface, the front-end hands the code over to the back-end and waits for the response from the auto-grader. When the front-end receives back the grading result, it records the attack detail to the database and visualize the result to the user.

The response is a fine-grained grading result with the following information: (1) success / failure rate, (2) the attacker advantage, and (3) average query count per attack, which are necessary for debugging and revising the solution in case of a failure, or improving the attack, *e.g.*, to achieve a higher advantage or to use less number of queries, in case the attack succeeded. An example of submission result is shown in Figure 4.

2.3 Back-end

The back-end plays an essential role in executing the user code in a safe environment, grade it, and generating a feedback. We designed the back-end components to reside in separate docker containers, such that even if the auto-grader or sandboxed executor is compromised, the impact is limited to manipulating the feedback of the submission of the malicious user.

Auto-grader. Auto-grader is the outermost module that handles the grading requests. It passes the submitted solution code to the sandboxed executor and provide the parameters, primitives, and evaluate the correctness and various metrics based on the challenge-specific test code.

Sandboxed executor. The sandboxed executor runs the user code in a very restricted and isolated environment to prevent malicious activities targeting the platform itself. User code are isolated not only with our system, but also isolated from each other to prevent manipulation of evaluation result or stealing the solution code.

Leveraging Google's NsJail for Linux namespace based process isolation, in addition to basic uid/gid drop, the sandboxed executor applies several restrictions: (1) disallows network connection, since it can lead to non-deterministic solution code, *e.g.*, receiving and executing code from a remote server, and potentially be abused for attacking the execution environment; (2) limits OS resource usage, *e.g.*, real time, CPU time, memory usage, to prevent DoS attacks; (3) mounts namespace and read-only filesystem to only expose essential files to the sandbox and prevent any modification attempts; (4) confines PID namespace to prevent information leak or DoS against other grading sessions; (5) confines UID namespace and `PRCTL_NO_NEW_PRIVILEGES` to eliminate the attack surfaces of SUID/SGID binaries, and (6) applies seccomp-bpf syscall filters to reduce the kernel attack surfaces. After all, the sandboxed process is allowed to communicate with the auto-grader only via gRPC and Unix domain socket based channel.

"Let $F : \{0,1\}^k \times \{0,1\}^m \longrightarrow \{0,1\}^n$ be a secure PRF. Consider $F1 : \{0,1\}^k \times \{0,1\}^{2m} \longrightarrow \{0,1\}^{2n}$ specified for all $x_1, x_2 \in \{0,1\}^m$ and all $K \in \{0,1\}^k$ by $F1(K, x_1 || x_2) = F(K, x_1) || F(K, x_2)$. Prove that $F1$ is not a secure PRF."

Figure 2: Problem 2.1.1, taken from Homework 2 of Spring 2020 CS6260 course.

3 IMPLEMENTATION

The prototype of Cryptomato is implemented[1] and running on a Ubuntu server machine with Intel Xeon Silver 4110 processor and 96 GB main memory. Cryptomato service is accessible at https://allspark.gtisc.gatech.edu:8080. The front-end is mainly written in TypeScript using Next.js framework and Prisma database toolkit. The back-end is mainly written in Python. It uses gRPC protocol for API endpoint specification and leverages Google's NsJail for sandboxing. Cryptomato supports running as orchestrated Docker containers; this design facilitates the deployment and the management of the service.

3.1 Working example

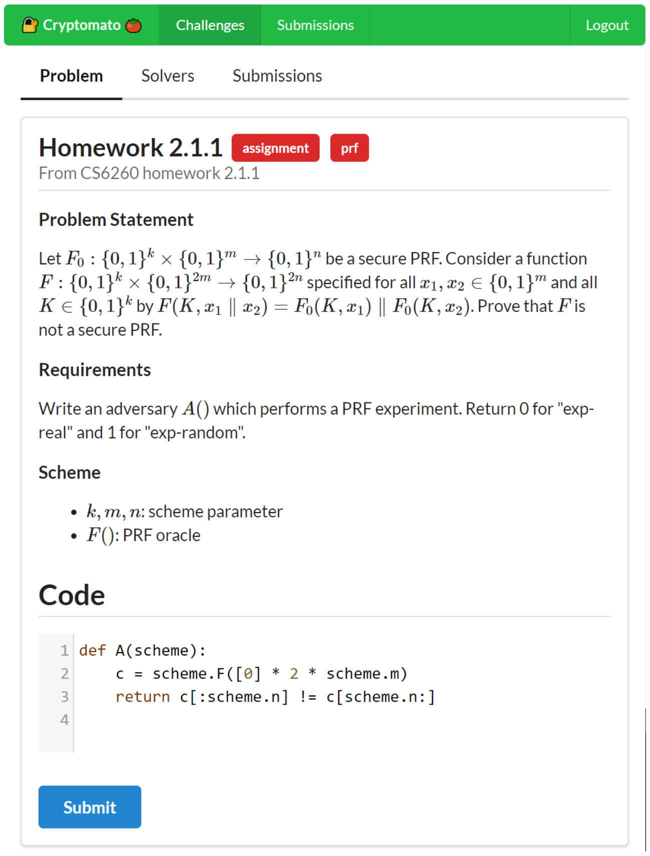
In this section, we walk through the workflow of Cryptomato with an example of solving Problem 2.1.1 that was given as a homework in CS6260 class.

Figure 2 is the original text excerpt from CS6260's Homework 2. This problem tests the ability of a student to formulate an adversarial attack, which renders the PRF-advantage not close to zero, within a reasonable time, and with reasonable number of queries of moderate lengths.

Figure 3 shows the same problem, transplanted as a challenge in Cryptomato. The problem category, description, and input box for solution code is shown to the students. Students can fill the input box with their attack code and submit it through the web interface.

Figure 5 is the actual ported code of problem 2.1.1, and Figure 6 is the grading function. Instructors can define functions and multiple parameters in advance to set boundaries or limit the resource. In this example, `scheme.F()` defines function F_1 given in the problem, which simply takes advantage of the `SecurePRF` primitive pre-defined by our Cryptomato module. Please note that tweaking the problem is very straightforward; the instructor only needs to define `F` that matches the intended definition.

[1]The source code of Cryptomato is publicly available at https://github.com/Qwaz/cryptomato



Homework 2.1.1
From CS6260 homework 2.1.1

Problem Statement

Let $F_0 : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^n$ be a secure PRF. Consider a function $F : \{0,1\}^k \times \{0,1\}^{2m} \to \{0,1\}^{2n}$ specified for all $x_1, x_2 \in \{0,1\}^m$ and all $K \in \{0,1\}^k$ by $F(K, x_1 \parallel x_2) = F_0(K, x_1) \parallel F_0(K, x_2)$. Prove that F is not a secure PRF.

Requirements

Write an adversary $A()$ which performs a PRF experiment. Return 0 for "exp-real" and 1 for "exp-random".

Scheme

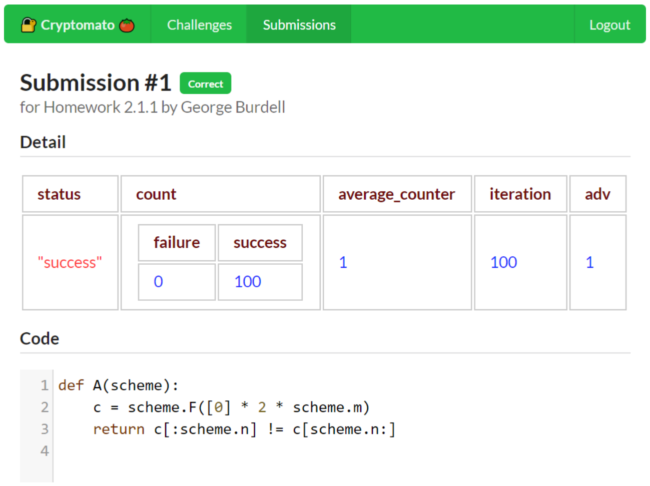
- k, m, n: scheme parameter
- $F()$: PRF oracle

Code

```
def A(scheme):
    c = scheme.F([0] * 2 * scheme.m)
    return c[:scheme.n] != c[scheme.n:]

```

Figure 3: The submission page for problem 2.1.1.



Submission #1
for Homework 2.1.1 by George Burdell

Detail

status	count		average_counter	iteration	adv
	failure	success			
"success"	0	100	1	100	1

Code

```
def A(scheme):
    c = scheme.F([0] * 2 * scheme.m)
    return c[:scheme.n] != c[scheme.n:]

```

Figure 4: An example report for user's attack on problem 2.1.1.

The code for a model solution is shown in Figure 7, which defines an attacker function A. The attack is equivalent to retrieving $c \leftarrow F(0 \parallel 0)$, and returning 0 (real) if $c[0] ==$ $c[1]$ or 1 (random) otherwise. Given this solution code, the `test` function in Figure 6 statistically computes the cpa-cg advantage by testing the attacker code 100 times and reports the result as a JSON object to the front-end. The front-end receives and parses the result object, and visualizes the result as shown in Figure 4.

Table 1: Challenges ported from CS6260 homework assignments and their topics.

#	Problem ID	Topic	Primitives
01	2.1.1	PRF	SecurePRF
02	2.1.2	PRF	SecurePRF
03	2.3	IND-CPA	SecurePRF
04	3.1	IND-CCA	SecurePRF
05	3.2	UF-CMA	SecurePRF
06	3.3	UF-CMA	SecurePRF
07	4.1	UF-CMA	SHA-3
08	5.6	Number theory	ElGamal
09	6.1.2	IND-CCA	ElGamal
10	6.1.3	Number theory	RSA
11	6.2	Weak parameter	ElGamal, SHA-1

Table 2: Result of testing 11 challenges with two correct solutions and two incorrect solutions, for each challenge.

	Correct solution	Incorrect solution
Success	22	0
Failure	0	22

4 EVALUATION

In this section, we evaluate the coverage, correctness, performance and advantages of CRYPTOMATO, and compare it with an existing solution.

4.1 Coverage of topics

Out of 24 problems given in the six homework assignments throughout the semester, 11 problems (Table 1) were suitable for porting to CRYPTOMATO. From the table, we can find that CRYPTOMATO supports testing of all relevant topics, including PRF, IND-CPA, IND-CCA, UF-CMA, number theory, and weak parameter.

4.2 Correctness

We evaluate the correctness of CRYPTOMATO in various scenarios, including the submission of a malicious code.

User code has syntax errors. The nature of CRYPTOMATO's code evaluation is dynamic; the submitted code is run within the sandboxed executor. Thus, syntax errors are returned to users without affecting the grading system.

User code does not terminate. The grader features a configurable timeout, which defaults to 30 seconds. This kills any non-terminating code, ensuring availability of CRYPTOMATO.

User code tries to exploit the system. A high-value, critical vulnerability in the gRPC library or Linux kernel could exist and fully compromise our system. However, a recent security audit of gRPC library demonstrated strong security of it [4]. Also, a Linux kernel vulnerability that could allow attackers to escape our sandbox has up to $50k market value [6], so we conclude that it is safe to assume such attack is unlikely to target educational platforms.

User code is incorrect. False positive in auto-grader is very undesirable, as wrong, incorrect answers could falsely reward students with positive scores. We tested the 11 challenges in Table 1 by manually writing two incorrect solutions for each challenge. For example, for testing problem 2.1.1, we implemented two different attacks that result in the attacker advantage 0, either by wrong attack, or mistake in the implementation. The result is shown in Table 2. CRYPTOMATO's auto-grader returned "failure" for all 22 incorrect solutions. More test cases might reveal corner cases, but this experiment proves that until now, the false positive rate is 0%.

User code is correct. Obviously, correct answers successfully passing the tests is as much important as having zero false-positives, and we evaluated the same 11 challenges with two correct answers each. CRYPTOMATO's auto-grader returned "success" for all 22 correct solutions (Table 2), ensuring the correctness of the grading system.

4.3 Performance

Quick feedback after the submission of code is a positive input that determines the quality of a learning experience. We evaluated the performance of CRYPTOMATO by measuring how much time is required for a student to get the feedback, using all 11 challenges, 22 correct and 22 incorrect solutions. The result is promising; with the default configuration of evaluating the solution for 100 times, all challenges except for 5.6 were graded within a second. Solutions for challenge 5.6 required about 10 seconds, because of the time complexity of generating parameters.

4.4 Instructor advantage

On top of the expected advantages of offloading the grading efforts to CRYPTOMATO, instructors can also benefit as tweaking existing challenges or creating new challenges is made easy. For example, by tweaking the `mid` variable in function `F1` of Figure 5, an entirely different challenge is made.

4.5 Comparison to RootMe

We also evaluate how CRYPTOMATO fares against the existing cryptography challenges of CTF war-games or security-study platforms with regards to learning and exercising cryptanalysis concepts. RootMe [5] is one of the most well-known platforms that has been offering the largest number and variety of practical content dedicated to cybersecurity for 10 years, with more than 300,000 users worldwide. Its cryptanalysis section presents 49 challenges about breaking encryption algorithms to decipher encoded data and find flags. Surprisingly, none of these 49 challenges require users to analyze the attacker advantage or resource. For example, for challenge *Encoding - ASCII*, a 61-byte hexadecimal string is given, and simply interpreting each byte into ASCII reveals the flag. Submitting this flag, user earns points as a reward and this challenge is marked complete. The other 48 challenges share similar concept and workflow. We must admit that such challenges can be entertaining and may interest students into cryptanalysis. However, as opposed to CRYPTOMATO, they can hardly serve an educational purpose that CS6260 needs.

5 CONCLUSION

In this report, we presented CRYPTOMATO, an interactive platform to assist the education of cryptographic schemes. By designing and implementing attacks by themselves, as well as receiving feedback about their attack instantly, students can go through an invaluable learning process, and cultivate essential skills for cryptanalysis, which is essential in the modern computing ecosystem. Instructors, using CRYPTOMATO, can avoid tedious work of manually grading submissions, and rather focus on creating new challenges. As future work, we plan to devise and add more challenges.

REFERENCES

[1] 2015. gRPC - A high-performance, open source universal RPC framework. https://grpc.io/. (2015).
[2] 2016. The HTTPS-Only Standard. https://https.cio.gov/. (2016).
[3] 2019. DEF CON 27. defcon.org/html/defcon-27/dc-27-ctf.html. (2019).
[4] 2019. gRPC Security Audit. https://github.com/grpc/grpc/blob/master/doc/security_audit.md. (2019).
[5] 2020. Root Me : Hacking and Information Security learning platform. https://www.root-me.org/. (2020).
[6] 2020. Zerodium - Our Exploit Acquisition Program. https://zerodium.com/program.html. (2020).
[7] Chase Bank. 2020. How We Protect You. chase.com/digital/resources/privacy-security/security/how-we-protect-you. (2020).
[8] The Register. 2019. 620 million accounts stolen from 16 hacked websites now for sale on dark web, seller boasts. https://www.theregister.co.uk/2019/02/11/620_million_hacked_accounts_dark_web/. (2019).
[9] The New York Times. 2018. Marriott Data Breach Is Traced to Chinese Hackers as U.S. Readies Crackdown on Beijing. https://www.nytimes.com/2018/12/11/us/politics/trump-china-trade.html. (2018).

APPENDIX A CHALLENGE, GRADING, AND SOLUTION CODE

```python
from cryptomato import *

class Scheme:
    """
    PRF: k-bits x m-bits => n-bits
    """
    exports = [
        'k', 'm', 'n',  # params
        'F'  # PRF setting
    ]

    def __init__(self, b, k, m, n):
        self.b = b
        self.k, self.m, self.n = k, m, n
        self.Fbase = SecurePRF(random_bits(k), m, n)
        self.g = SecurePRF(random_bits(k), m, n)
        self.counter = 0

    @counted
    def F1(self, m):
        assert self.b in (0, 1)

        if self.b == 1:
            return self.g(m)
        else:
            # Problem 2.1.1
            assert len(m) % 2 == 0
            mid = len(m) // 2
            return self.Fbase(m[:mid]) + self.Fbase(m[mid:])
```

Figure 5: Problem 2.1.1, ported for Cryptomato.

```python
def test(module): # PRF tester
    counter = [0]
    ITERATION = 100
    res = [0, 0]

    for i in range(ITERATION):
        k, m, n = 64, 64, 64
        b = random.getrandbits(1)
        scheme = Scheme(b, k, m, n)
        res[module.A(scheme) == b] += 1

    failure, success = res
    adv = success / ITERATION

    return {
        "status": "success" if adv > 0.6 else "failure",
        "count": {
            "failure": failure,
            "success": success
        },
        "average_counter": counter[0] / ITERATION,
        "iteration": ITERATION,
        "adv": adv
    }
```

Figure 6: PRF advantage and resource tester functionality for problem 2.1.1.

```python
def A(scheme):
    c = scheme.F([0] * 2 * scheme.m)
    return c[:scheme.n] != c[scheme.n:]
```

Figure 7: Model solution for problem 2.1.1.