This document describes the per-question codes used in the **"They're not that hard to mitigate": What Cryptographic Library Developers Think About Timing Attacks** paper. The format is the following:

```
{question_code}: {question_text}
 - {code_name}: {code_description}
 - {code_name}: {code_description}
 - ...
```

Some questions which were aimed at the library and not at the person were coded by coding the individual responses of the libraries developers and then merging the codes into one code per library as described in the paper. We mark the library-level codes with `[library]` and the individual with `[person]`.

Codebook

LibraryDevelopmentRole [person]: What's your role in the development of {library/primitive}? (we code mostly self-descriptions; if we don't have self-description, we code according to description in brackets)

- all roles - single person: There is only one person that does all the things.
- project lead: Decides on project direction.
- core developer: Works on essential parts of the project, sometimes this is an officially defined role.
- maintainer: Fixes bugs, issues releases and maintains the library or its part.
- committer: Has commit rights.
- contributor: Contributes but does not have commit rights.
- not involved: Is not involved with the development.

LibraryDesignDecisions: How are you involved in design decisions (e.g., concerning the API, coding guidelines and style, security-relevant properties) for {library/primitive}?

- Participant [person]
    - involved
    - not involved
- Process [library]
    - discussion: The project discusses design decisions.
    - voting: The project votes on design decisions.
    - consensus: The project needs to reach unanimous agreement on design decisions.
    - final say: The project has a person with a final say for (a part of) its design decisions.

LibraryUseCases [library]: What are the intended use cases of {library/primitive}?

- Platform [library]
    - server: big processors using a common architecture, throughput relevant.
    - desktop: common processors for developers.
    - embedded device: with OS, 32bit, processors less common for developers.
    - mobile: power consumption is also a factor.
    - micro-controller: no OS, 8/16bit, very constrained resources, performance less important.

    - TEE: Trusted Execution Environment, a special part in processors mainly under control of the manufacturer.
- Target [library]
    - TLS: web, also others.
    - protocols: may include TLS but is not limited to that.
    - services: throughput is important.
    - cloud: untrusted environment.
    - operating system: used in operating system for all kinds of tools.
    - crypto-currency: High monetary values.
    - corporate internal: Less security critical environment.
    - teaching: Code has to be simpler, readable, protections may inhibit readability.
    - TEE: The processor manufacturer is involved in making the code usable for the end users.

LibraryThreatModel [library]: What is the threat model for {library/primitive} with regards to side-channel attacks?

- Remote: Considers remote attacks to be in the threat model.
    - yes
    - no
    - if easy
- Local: Considers local attacks to be in the threat model.
    - yes
    - no
    - if easy
- Speculative: Considers speculative execution attacks to be in the threat model.
    - yes
    - no
    - if easy
- Physical: Considers physical (not fault injection) attacks to be in the threat model.
    - yes
    - no
    - if easy
- Fault: Considers fault injection attacks to be in the threat model.
    - yes
    - no
    - if easy
- Best effort: “protecting against attacks perceived as most threatening”
- Don’t know: Doesn't know what the threat model is.

TimingAttacksRelevant [library]: Do you consider timing attacks a relevant threat for the intended use of {library/primitive} and its threat model? Please give a brief explanation for why / why not.

- Yes
    - It is easy/not hard to do
    - threats of key-recovery in asymmetric crypto
    - user demands
    - fear of loss of reputation
    - hostile environment
    - attacks get smarter

    - (rising) relevance (important for reason, use case, etc.)
    - personal expectation (person directly states ...)
    - connected environment
    - no explicit reason
    - large scope (large group of people)
- No
    - not the goal
    - not really - only “practical” attacks

TimingAttacksResistant [library]: Does {library/primitive} claim resistance against timing attacks? *//Single-choice question but included for completeness.//*

- Yes
- No
- Don't know
- Partially
- No, but planning to

HowDecide [library]: How did the development team decide to protect or not to protect against timing attacks?

- Personally: One person decided.
- Team decision: Decision after discussion, potentially voting.
- Don't know
- Corporate decision: Executives decided.
- Priority trade-off: There was a trade-off considered during the decision making on the priority of tasks given limited resources.
- Obvious: It was obvious that protection was necessary.
- Inherited: The project inherited the protection, from a previous project or development team.

HowProtect [library]: How does {library/primitive} protect against timing attacks?

- Hardware: Uses hardware features like AES-NI.
- Constant-time code practice: Uses the constant-time code practice to avoid leaks.
- Algorithm choice: Chooses algorithms that are known to be constant-time.
- Blinding: Randomizes values to prevent exploitation of leaks.
- Slicing: Uses a sliced implementation.
- Choice of module: The library contains multiple implementations of a part of it, only some of which might be constant-time.
- Assembly: Uses specialized low-level implementation for leak sensitive parts.
- In progress: The library strives to be constant-time but its protections are in progress.
- Random delays: Uses random delays to mask timing leaks.

TestLibrary [library, person]: Did you personally test for or verify the resistance of {library/primitive} against timing attacks? *//Single-choice question but included for completeness.//*

- Yes
- No
- Partially
- Not me but someone did
- Not yet but planning to
- I don’t know

HowTest [library, person]: How did you test or verify the resistance against timing attacks?

- Manual: Developer manually analyzed at least parts of the code or binary.
    - source code: The source code was analyzed for timing leaks.
    - binary: The compiled binary was analyzed for timing leaks.
    - statistics: The code was executed and its runtime measured to detect timing leaks.
    - runtime: The binary was run and analyzed with a debugger to check for timing leaks.
- Tool: Some tool was used in a (semi-)automated way.
    - valgrind
    - TriggerFlow
    - DATA
    - ctgrind
    - memsan
    - statistics

ToolUseProccess [person]: Please describe the process of using the tools.

- Worked (at least once, but not necessarily repeatedly)
- Didn't work
    - too much resource usage (effort, time, RAM, CPU cores, machines, ...)
- automatically (CI...)
- manually
    - during (development / before it's done)
    - after (development / it's done)

WhyNotTool [person]: Why have you not tried to use these?

- Lack of time
- Tool not available (they couldn't get to the tools sources)
- Tool not maintained (compiler or other dependency changes might break it)
- Uses other tool
- Did not work (set it up but it did not work after)
- Did not need
- Setup problems (even if a tool might have worked if more time was invested)
- Unable to ignore known issues (issues found in the analyzed program)
- Doesn't work for language/environment (f.e. tool only works for one programming language/compiler etc.)
- Confused (the answer is either inconsistent in itself, with their other answers, or has uncommon/wrong technical assumptions which others did not have)
- Did not know this tool (implies interest in the tool?)Tool uses too much resources (time, RAM, CPU cores, machines, ...)
- Source code changes necessary (markup for secret/public values, memory regions/aliasing, additional header files, ...)

Hypothetical tool use:

- Requirements
    - resources (they don't have enough time/computing power)
    - effort (they mentioned that setting up and using the tool is a significant effort)
        - annotations (they mentioned that code annotations require significant effort to create/maintain)
            - creating
            - maintaining

        - maintenance (they mentioned that maintaining the tool integration requires significant effort)
        - analysis (they mentioned that analysis of the tools results was a burden)
    - environment
        - not applicable (the tool cannot be applied due to the environment/language they work in)
        - dedicated (the tool requires a dedicated machine that doesn't have runtime noise to work in)
    - Guarantees limited (they mentioned that the tool offers limited guarantees and has either false positives or negatives)
    - concern
    - future maintenance of the tool (they are concerned that the tool developers will not maintain the tool in the future)
    - already use (they already use the tool somewhere)
    - already annotated (library has some form of secrecy annotation, so a tool is expected to use that)
    - will try (they will attempt to use the tool)
    - out of scope
    - lack of knowledge (about certain tools and their differences; might be misconceptions)