# Cryptographic Cryptid Protocols
## How to Play Cryptid with Cheaters

**Abstract.** Cryptid is a board game in which the goal is to be the first player to locate the cryptid, a legendary creature, on a map. Each player knows a secret clue as to which cell on the map contains the cryptid. Players take it in turns to ask each other if the cryptid could be on a given cell according to their clue, until one of them guesses the cryptid cell. This game is great fun, but completely loses its interest if one of the players cheats by answering the questions incorrectly. For example, if a player answers negatively on the cryptid cell, the game continues for a long time until all the cells have been tested, and ends without a winner. We provide cryptographic protocols to prevent cheating in Cryptid. The main idea is to use encryption to commit the players' clues, enabling them to show that they are answering correctly in accordance with their clue using zero-knowledge proofs. We give a security model which captures soundness (a player cannot cheat) and confidentiality (the protocol does not leak more information than the players' answers about their clues), and prove the security of our protocols in this model. We also analyze the practical efficiency of our protocols, based on an implementation of the main algorithms in Rust. Finally, we extend our protocols to ensure that the game designer has correctly constructed the cryptid games, *i.e.*, that the clues are well formed and converge on at least one cell.

**Keywords:** Provable security · Board game · Cryptid · Zero Knowledge Proof

## 1 Introduction

Cryptid is a deduction game based on the theme of cryptozoology: players are in search of a cryptid, a legendary creature, and try throughout the game to glean information to locate it on a map. The map is divided into cells, each with its own type (forest, desert, etc.) and several properties (close to a hut, animal territory, etc.). Each player receives a secret clue as to where the cryptid is located, which can take one of two forms: the cell of the cryptid is one of several given types, or the cell is at a given distance from something. The players then take it in turns to ask each other whether a given cell corresponds to their secret clue or not. As the game progresses, each player accumulates more and more information about the cryptid's cell, until one player deduces it and wins the game.

First released in 2018, the game has since enjoyed international success with a great reception from the player community and has been translated into multiple languages, which has earned it numerous festival nominations and even an award (Palme d'Or 2020 for Best Family Game). An online version of the game is

also available for free on `https://www.playcryptid.com/`. However, the game loses all interest if one of the players lies. Accumulating and processing information involves a considerable mental effort, and discovering that the game has been spoiled because a player has answered incorrectly is very frustrating. For instance, if a cheater answers negatively on the cryptid's cell, the game can last until all the cells have been invalidated before the other players become aware of the cheating, which is long. Worse still, the cheater can get away with winning the game without being discovered by the others, giving them a substantial advantage and encouraging dishonest players to cheat. Aware of the problem, the authors of the game have included a paragraph on honesty in the rulebook:

> "*Cryptid allows room for misdirection, but the game is completely dependent on all players [answering] honestly. If that doesn't sound like your gaming group, then this might not be the game for you.*"

It can be interpreted as an open problem left to cryptographers. In this paper we address this open problem by proposing a cryptographic version of cryptid to prevent cheating. We also extend the question to cases where the game designer is not honest and tries to give an advantage to one player or to design games where no player can win.

*Motivation.* Our work offers the possibility to play online such that players no longer have to trust other players, the game designer, or the server that manages the games. In addition to this direct application, our motivation is also pedagogical. Cryptography makes it possible to secure protocols in which entities interact with sensitive data, even if they do not behave as expected. The game analogy illustrates this concept: Entities play a game in which each player has secret values, and cryptography ensures that everyone respects the rules of the game (*i.e.,* does not cheat) without exposing the secret values. The playful nature of the games makes it easier to interest the general public in these topics, and to introduce counterintuitive concepts such as zero-knowledge proofs. Discovering these powerful tools through fun applications then opens the way to presenting similar protocols with more concrete societal applications, such as e-voting [10] or e-cash [8]. Although the general public will be able to understand the concepts used to secure Cryptid, they will not necessarily have the knowledge to understand the technical part of our work, but it can attract and stimulate a more experienced audience (*e.g.,* computer science students who enjoy playing board games) to discover cryptographic protocols, security models, and security proofs through a playful and relatable example.

*Contributions.* We begin by giving a formal treatment to the cryptid game and its security. We define what a cryptographic cryptid protocol is and model two security properties: soundness, which ensures that a player cannot lie when answering a question, and confidentiality, which ensures that players learn nothing more than what they are supposed to know according to the rules of the game through the protocol. We then propose two instantiations of this protocol based on two ways of encoding the clues. The first requires a large storage capacity

for each clue (linear in the number of cells, *i.e.,* 108 in the original game) but little computational power from the players, while the second requires moderate storage capacity and computational power (linear in the number of properties a cell can have, *i.e.,* 14 in the original game).

We then address the case where the game master (who constructs the maps and clues) is not honest. Indeed, the game must be designed in such a way that the clues respect the structure given in the game rules (which allows the clues to be deduced from the answers to the questions), and that the clues converge on at least one cell on the board (which ensures that the game can end when this cell is discovered). In the physical version of the game, players cannot verify this without knowing each other's clues. So they have to trust the game designers, who may have built impossible games to confuse them. Thus, we extend our model and our second protocol to take into account security against malicious game masters.

Our three protocols use the same paradigm: the clues are encrypted by the game master, then we use zero-knowledge proofs on these encrypted clues to allow the master to prove that the game is well designed and to allow the users to prove they answer correctly. For each protocol, we prove the security properties by reduction in our security model, under the assumption that the encryption scheme we use is IND-CPA and that the proofs are zero-knowledge and sound. The third protocol additionally requires the use of a partially homomorphic encryption scheme (we use ElGamal) and the hash function proposed in [5], which is collision resistant under the discrete logarithm assumption. We also provide a Rust implementation of the main algorithms used in our protocol, then we evaluate in practice the size of the cryptographic data generated and exchanged during a game and the computation time required to produce them.

*Related Works.* The first cryptographic protocol to enable a secure electronic version of a board game was mental poker, proposed by Shamir, Rivest, and Adelman [20]. This protocol enables remote users to play poker with the same properties as a classic card game. Numerous works have extended this result to card games in general [1,16,19,22], proposing different security models. However, these protocols make it possible to play online games that are usually physical, guaranteeing only the same properties as those of the physical version.

Another line of work focuses specifically on trick-taking games (like Whist or Bridge), and guarantees more security than the physical version of the game [2,6]: in trick-taking games, a player cannot always play any of the cards in their hand, and if the player decides to cheat, the cheating will only be detected much later, when the player plays a card they are not supposed to have. Cryptographic trick-taking game protocols force the player to prove that they are playing correctly without leaking any information about the player's cards, thus preventing cases of cheating that would not have been detected immediately with real cards. Our work has a similar objective since we aim to play Cryptid while avoiding the cheating problems that are unavoidable in the physical version, but concerns a game with a very different structure.

Finally, we could have used generic tools to easily reach our goal, such as multi-party computation protocols [12] or zero-knowledge proofs for circuits [15]. However, these tools are heavy due to their genericity (in short, they require the implementation of the function to be evaluated by a boolean or arithmetic circuit, and the evaluation of each of the operations of this circuit on encrypted data), and would have been much less efficient than our solutions.

## 2 Rules of Cryptid

We briefly explain the rules of the Cryptid game. A full version of the rules, including a description of the physical material of the game, can be found in Appendix A.

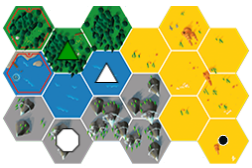


**Fig. 1.** A section of a cryptid map.

Cryptid is played on a map consisting of hexagonal tiles called "*cells*". Each cell has a type among "*forest*", "*mountain*", "*swamp*", "*water*", and "*desert*", represented by a color. In addition to its type, a cell may contain a "*bear territory*" (the cell is circled in dotted lines), a "*cougar territory*" (the cell is circled in red), and/or a "*structure*". A structure has one color and can be an "*abandoned shack*" (represented by a triangle) or a "*standing stone*" (represented by an octagon). For instance, Figure 1 shows a portion of the map with "*forest*" cells in green, "*desert*" cells in yellow, "*mountain*" cells in gray, and "*water*" cells in blue. The two cells at the top left contain a cougar territory. Three cells contain structures: a white standing stone, a white abandoned shack, and a green abandoned shack.

Each player is secretly given a clue. This clue can take two forms:

- the type of cell in which the cryptid is located is one of two given types (there are therefore $10 = \binom{5}{2}$ such clues),
- the cell where the cryptid is located is $n$ cells away from a cell with a specific property.

In the second case, there are 14 different possible clues:

- one cell or less from a cell of a given type (among 5), or of the territory of any animal (6 cases)
- two cells or less from a stone, a shack, a cougar territory, or a bear territory (4 cases)
- three cells or less from a blue, white, green, or black structure (4 cases).

The game also offers a *difficult mode* in which clues can be the negation of one of the 20 clues mentioned above (we do not consider the difficult mode in this paper and leave it for future works).

For the sake of formalism, we consider these 14 neighborhood properties as cell properties directly. For example, the cell at bottom left in Figure 1 is considered to have the 8 following neighborhood properties: one cell from a mountain, water, and animal territory, two cells from a stone, shack, and cougar territory, and three cells from a white and green structure.

Each player in turn asks another player a question as follows: the player points to a cell, and ask one other player to say whether that cell could be the cryptid's or not, according to its clue. When the player thinks they know which cell the cryptid is on, they can instead ask all the other players to take turns answering on that cell. If all players answer affirmatively, then the player has found the cryptid cell and wins the game. However, after each negative answer, the player must leak a little of their clue by pointing to another cell that has never been invalidated and revealing that it cannot be the cryptid's cell.

In its physical version, the game contains material to build 180 maps of 108 cells for 3 to 5 players. With a booklet system, each player can secretly find their clue for a given map. The 180 game setups are constructed in such a way that only one of the cells corresponds to the clues of all players. Note that to ensure that each game ends correctly, it is sufficient to ensure that at least one cell matches all player's clues.

## 3 Cryptographic Background

First of all, here are the notations we will be using throughout this paper. We denote by $[\![n]\!]$ the set $\{0,\dots,n\}$ and $[\![n]\!]^*$ the set $\{1,\dots,n\}$ where $n \in \mathbb{N}$. By $x \leftarrow y$ we mean that the variable $x$ takes a value $y$, by $x \leftarrow \mathsf{Algo}(y)$ that the variable $x$ takes a value outputted by the algorithm $\mathsf{Algo}$ on input $y$, and by $r \xleftarrow{\$} S$ that $r$ is chosen from the uniform distribution on $S$. We use the acronym p.p.t. for *probabilistic polynomial time*, and s.t. for *such that*.

We now recall the definition of negligible function, public-key encryption scheme, ElGmal encryption scheme, and non-interactive proof systems.

**Definition 1 (Negligible function).** *A function $\epsilon : \mathbb{N} \to \mathbb{R}$ is said to be negligible if for any positive polynomial $t$, there exists an integer $n$ s.t. for all integer $x > n$, $|\epsilon(x)| < \frac{1}{t(x)}$.*

**Definition 2 (Public-key encryption scheme).** *A public-key encryption scheme $\mathcal{E}_\lambda$ is a tuple of three algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows:*

$\mathsf{Gen}(\lambda)$: *takes as input a security parameter $\lambda$ and returns a public/private key pair $(\mathsf{pk}, \mathsf{sk})$.*

$\mathsf{Enc}_{\mathsf{pk}}(m)$: *takes as input the public key $\mathsf{pk}$ and a message $m$, and returns a ciphertext $c$.*

$\mathsf{Dec}_{\mathsf{sk}}(c)$: *takes as input the secret key $\mathsf{sk}$ and a ciphertext $c$, and returns a message $m$.*

*The IND-CPA experiment $\mathit{Exp}^{\mathit{IND\text{-}CPA}}_{\mathcal{E}_\lambda,\mathcal{A},b}(\lambda)$ of a public-key encryption $\mathcal{E}_\lambda$ is defined as follows, where $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is a pair of p.p.t. algorithms: the experiment generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\lambda)$, runs $(m_0, m_1) \leftarrow \mathcal{A}_1(\mathsf{pk})$, computes $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b)$, runs $b' \leftarrow \mathcal{A}_2(c)$, and returns $b = b'$. $\mathcal{E}_\lambda$ is said to be IND-CPA if for any pair of p.p.t. algorithms $\mathcal{A}$, there exists a negligible function $\epsilon_{\mathit{IND\text{-}CPA}}$ s.t.:*

$$\left|\Pr\left[0 \leftarrow \mathit{Exp}^{\mathit{IND\text{-}CPA}}_{\mathcal{E}_\lambda,\mathcal{A},0}(\lambda)\right] - \Pr\left[1 \leftarrow \mathit{Exp}^{\mathit{IND\text{-}CPA}}_{\mathcal{E}_\lambda,\mathcal{A},1}(\lambda)\right]\right| \leq \epsilon_{\mathit{IND\text{-}CPA}}(\lambda).$$

**Definition 3 (ElGamal encryption scheme).** *Let $\lambda$ be a security parameter. The ElGamal encryption $\mathit{ElGamal} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ instanciated by a group $\mathbb{G}$ of prime order $p$ generated by $g$ s.t. $p > 2^\lambda$ is defined as follows:*

$\mathsf{Gen}(\lambda)$: *picks* $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$, *computes* $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$, *and returns* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Enc}_{\mathsf{pk}}(m)$: *picks* $r \xleftarrow{\$} \mathbb{Z}_p^*$, *computes* $c_1 \leftarrow g^r$, *computes* $c_2 = \mathsf{pk}^r m$, *and returns* $c = (c_1, c_2)$.

$\mathsf{Dec}_{\mathsf{sk}}(c)$: *computes* $m \leftarrow c_2/(c_1^{\mathsf{sk}})$ *and returns it.*

The ElGamal encryption is known to be IND-CPA [21] under the Decisional Diffie-Hellman assumption [4]. Moreover, by defining the binary operation $\cdot$ between two ciphertexts $c = (c_1, c_2)$ and $c' = (c'_1, c'_2)$ as $c \cdot c' = (c_1 c'_1, c_2 c'_2)$, the following property holds: if $m = \mathsf{Dec}_{\mathsf{sk}}(c)$ and $m' = \mathsf{Dec}_{\mathsf{sk}}(c')$, then $mm' = \mathsf{Dec}_{\mathsf{sk}}(c \cdot c')$. ElGamal is said to be partially homomorphic.

**Definition 4 (Non-Interactive Proof system (NIP) [3]).** *Let $\mathcal{R}$ be a binary relation and $\mathcal{L}$ a language depending on a security parameter $\lambda$ s.t. $s \in \mathcal{L} \Leftrightarrow (\exists\, w, (s, w) \in \mathcal{R})$. A* Non-Interactive Proof system (NIP) *for the language $\mathcal{L}$ is a couple of algorithms* $(\mathsf{NIP}, \mathsf{Ver})$ *defined as follows:*

$\mathsf{NIP}\,\{w : (s, w) \in \mathcal{R}\}$: *takes as input a witness $w$ and a statement $s$ s.t. $(s, w) \in \mathcal{R}$, and returns a proof $\pi$.*

$\mathsf{Ver}(s, \pi)$: *takes as input a statement $s$ and a proof $\pi$, returns an acceptance bit $b$.*

*A* NIP *has the following security properties:*

**Soundness.** A NIP is *sound* if there is no p.p.t. algorithm $\mathcal{A}$ s.t. $\mathcal{A}(\mathcal{L})$ outputs $(s, \pi)$ s.t. $\mathsf{Ver}(s, \pi) = 1$ and $s \notin \mathcal{L}$ with non-negligible probability.

**Extractability.** A NIP is *extractable* if for any algorithm $\mathcal{A}$ and any statement $s$ s.t. $\mathcal{A}(\mathcal{L})$ outputs $\pi$ s.t. $\mathsf{Ver}(s, \pi) = 1$ with non-negligible probability, there exists a negligible function $\epsilon$ and a p.p.t. algorithm Ext having access to $\mathcal{A}(\mathcal{L})$ as an oracle that outputs $w$ s.t. $(s, w) \in \mathcal{R}$ with probability $1 - \epsilon(\lambda)$.

**Zero-knowledge.** A NIP is *zero-knowledge* if the proof $\pi$ leaks no information, *i.e.*, there exists a p.p.t. algorithm Sim (called the simulator) s.t. for any $(s, w) \in \mathcal{R}$, the outputs of the algorithms $\mathsf{NIP}\{w : (s, w) \in \mathcal{R}\}$ and $\mathrm{Sim}(s)$ follow the same distribution.

## 4 Cryptographic Cryptid

### 4.1 Formal Definitions

We begin by giving a formal treatment of a cryptid game, defining the required algorithms and elements. We define the clues, the map and the algorithm for answering a question on a cell of the map in accordance with a clue.

**Definition 5 (Cryptid Encoding).** *A* cryptid encoding *is a tuple* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$ *where $n$ is an integer called the* number of players, *$\mathcal{C}$ is a set called the* clue set, *$N$ is an integer called the* size of the map, *$\mathcal{M}$ is a set called the* map set *(where each* map $\in \mathcal{M}$ *is a vector of $N$ elements* map $= (\mathrm{cell}_i)_{i \in [\![N]\!]^*}$ *and*

*each* $\text{cell}_i$ *is called a* cell *of the* map*), and* Answer *is a p.p.t. algorithm defined as follows:*

$\mathsf{Answer}(\text{map}, \text{clue}, j)$**:** *takes as input a map* $\text{map} \in \mathcal{M}$*, a clue* $\text{clue} \in \mathcal{C}$ *and an index* $j \leq N$ *of a cell. It returns* $A \in \{\texttt{maybe}, \texttt{no}\}$ *the correct answer for the cell* $j$ *according to the rules of Cryptid ("*`maybe`*" corresponds to bit* 1 *and "*`no`*" to* 0*).*

We define a cryptographic cryptid as a tuple of algorithms. In particular, a cryptographic cryptid contains an algorithm GenClue for committing a clue, and an algorithm Play for proving that the answer of a player on a cell is correct according to the committed clue. These algorithms can be used in a cryptid game to provide some security properties.

**Definition 6 (Cryptographic Cryptid).** *A* cryptographic cryptid *for a cryptid encoding* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$ *is a tuple of p.p.t. algorithms* $(\mathsf{Setup}, \mathsf{GenClue}, \mathsf{OpenClue}, \mathsf{Play}, \mathsf{Verify})$ *defined as follows:*

$\mathsf{Setup}(\lambda)$**:** *takes as input the security parameter* $\lambda$ *and returns a setup* set*. This setup is implicitly used as input in the other algorithms.*

$\mathsf{GenClue}(\text{clue})$**:** *takes as input* $\text{clue} \in \mathcal{C}$ *and returns the pair of public/secret clue keys* $(\mathsf{pc}, \mathsf{sc})$*.*

$\mathsf{OpenClue}(\mathsf{pc}, \mathsf{sc})$**:** *takes as input the pair of public/private clue key* $(\mathsf{pc}, \mathsf{sc})$ *and returns* $\text{clue} \in \mathcal{C}$*.*

$\mathsf{Play}(\mathsf{pc}, \mathsf{sc}, \text{map}, j, A)$**:** *takes as input a public/private clue key pair* $(\mathsf{pc}, \mathsf{sc})$*, a map* $\text{map} \in \mathcal{M}$*, an index* $j$*, and an answer* $A \in \{0, 1\}$ *for the cell* $j$ *in* map*. It returns a proof* $\pi$*.*

$\mathsf{Verify}(\pi, \mathsf{pc}, \text{map}, j, A)$**:** *takes as input a proof* $\pi$*, a public clue key* pc*, a map* $\text{map} \in \mathcal{M}$*, an index* $j$*, an answer* $A$*, and returns* $b \in \{\texttt{accept}, \texttt{reject}\}$ *("*`accept`*" corresponds to bit* 1 *and "*`reject`*" to* 0*).*

For playing the game using cryptographic algorithms, the players use the following protocol.

**Definition 7 (Cryptid Protocol).** *Let* $\lambda$ *be a security parameter. The* cryptid protocol *for* $n$ *players is a protocol between* $n + 1$ *parties, called the players (denoted* $\text{Player}_\alpha$ *for* $\alpha \in [\![n]\!]^*$*) and the game master (denoted* Master*). This protocol is instantiated by a cryptid encoding* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$*, a cryptographic cryptid* $(\mathsf{Setup}, \mathsf{GenClue}, \mathsf{OpenClue}, \mathsf{Play}, \mathsf{Verify})$*, a map* $\text{map} \in \mathcal{M}$*, and* $n$ *clues (denoted* $\text{clue}_\alpha \in \mathcal{C}$ *for* $\alpha \in [\![n]\!]^*$*). This protocol is defined as follows (where the setup* set *used has been generated by the algorithm* $\mathsf{Setup}(\lambda)$*).*

$\mathsf{Cryptid}\left\langle \{\text{Player}_\alpha(\text{set}, \text{map})\}_{\alpha \in [\![n]\!]^*}, \text{Master}(\text{set}, \text{map}, (\text{clue}_\alpha)_{\alpha \in [\![n]\!]^*}) \right\rangle$ :

- *For each* $\alpha \in [\![n]\!]^*$*,* Master *runs* $(\mathsf{pc}_\alpha, \mathsf{sc}_\alpha) \leftarrow \mathsf{GenClue}(\text{set}, \text{clue}_\alpha)$*.* Master *then sends* $\mathsf{sc}_\alpha$*, and* $(\mathsf{pc}_{\alpha'})_{\alpha' \in [\![n]\!]^*}$ *to each* $\text{Player}_\alpha$ *where* $\alpha \in [\![n]\!]^*$*.*
- *Each player* $\text{Player}_\alpha$ *computes* $\text{clue}_\alpha \leftarrow \mathsf{OpenClue}(\mathsf{pc}_\alpha, \mathsf{sc}_\alpha)$*.*
- *Players then play among themselves following ther rules. At the* $i^{th}$ *play:*
  - *One* $\text{Player}_{\alpha_i}$ *for some* $\alpha_i \in [\![n]\!]^*$ *receives an index* $j < N$ *from another player.* $\text{Player}_{\alpha_i}$ *runs* $A_i \leftarrow \mathsf{Answer}(\text{clue}_{\alpha_i}, \text{map}, j_i)$*, and* $\pi_i \leftarrow \mathsf{Play}(\mathsf{pc}_{\alpha_i}, \mathsf{sc}_{\alpha_i}, \text{map}, j_i, A_i)$*, then sends* $(A_i, \pi_i)$ *to everyone.*

- *For each* $\alpha \neq \alpha_i$, $\text{Player}_\alpha$ *checks that* $\mathsf{Verify}(\pi_i, \mathsf{pc}_{\alpha_i}, \text{map}, j_i, A_i) = 1$ *(if not,* $\text{Player}_\alpha$ *aborts the protocol).*

– *At the end of the game, each* $\text{Player}_\alpha$ *for* $\alpha \in [\![n]\!]^*$ *returns* $\mathsf{view}_\alpha = (\text{clue}_\alpha, \mathsf{sc}_\alpha, (\mathsf{pc}_{\alpha'})_{\alpha' \in [\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i \in [\![t]\!]^*})$*, where* $t$ *is the number of plays.* Master *returns nothing.*

We now formally define the two required security properties, namely soundness and confidentiality. Soundness must ensure that players cannot cheat, *i.e.,* that they cannot make a valid proof $\pi$ if they do not answer a question correctly.

**Definition 8 (Soundness).** *Let* $\lambda$ *be a security parameter and* $\Pi = (\mathsf{Setup}, \mathsf{GenClue}, \mathsf{OpenClue}, \mathsf{Play}, \mathsf{Verify})$ *be a cryptographic cryptid for a cryptid encoding* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$*.* $\Pi$ *is said to be* sound *if for any pair of p.p.t. algorithms* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the probability of success of the following experiment is negligible in* $\lambda$*:*

$\mathbf{Exp}^{\mathsf{Soundness}}_{\Pi,\mathcal{A}}(\lambda)$*:*
$\text{set} \leftarrow \mathsf{Setup}(\lambda)$
$\text{clue} \leftarrow \mathcal{A}_1(\text{set})$
$(\mathsf{pc}, \mathsf{sc}) \leftarrow \mathsf{GenClue}(\text{clue})$
$(\pi, \text{map}, j, A) \leftarrow \mathcal{A}_2(\mathsf{pc}, \mathsf{sc})$
$A' \leftarrow \mathsf{Answer}(\text{map}, \text{clue}, j)$
*return* $((\text{clue} \in \mathcal{C}) \wedge (\text{map} \in \mathcal{M}) \wedge (A' \neq \bot)$
$\wedge (A = 1 - A') \wedge \mathsf{Verify}(\pi, \mathsf{pc}, \text{map}, j, A))$

Confidentiality ensures the protection of players' clues. More precisely, an adversary must not be able to deduce any more information about players' clues in the cryptographic version than in the physical version of Cryptid. Note that each answer leaks some information about the secret clue; this leakage is inherent to the game and must be taken into account in our definition. Thus, confidentiality must ensure that, at any point in the game, a player has no more information about the clues of others than can be deduced from their answers.

We propose a simulation-based security definition [17]: a cryptographic cryptid is confidential if there exists a polynomial-time simulator that simulates for a player the cryptid protocol until play $t$, using only the values known by that player and the answers of the other players until play $t$. So, as the player could have simulated the whole protocol with their own data, we show that they learn nothing more than they already know during the game.

**Definition 9 (Confidentiality).** *Let* $\lambda$ *be a security parameter and* $\Pi = (\mathsf{Setup}, \mathsf{GenClue}, \mathsf{OpenClue}, \mathsf{Play}, \mathsf{Verify})$ *be a cryptographic cryptid for a cryptid encoding* $\mathsf{CE} = (n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$*. Let* $\mathsf{Cryptid}_{\alpha^*,t}$ *be a protocol defined as the cryptid protocol instantiated by* $\Pi$ *and* $\mathsf{CE}$ *(Definition 7) except that it aborts after* $t$ *plays and returns* $\mathsf{view}_{\alpha^*}$ *only.*

*We define a cryptid simulator* Sim *instantiated by* $\Pi$ *and* $\mathsf{CE}$ *as an algorithm that takes as input a setup* set*, an integer* $t$*, a map* $\text{map} \in \mathcal{M}$*, an integer* $\alpha^* \in [\![n]\!]^*$*, a clue* $\text{clue} \in \mathcal{C}$*, and a set of* $t$ *triplets* $\{(\alpha_i, A_i, j_i)\}_{i \in [\![t]\!]^*}$ *each containing*

*an integer* $\alpha_i \in [\![n]\!]^*$*, a bit* $A_i$ *and an integer* $j_i \in [\![N]\!]^*$*, and that returns a tuple* $(\mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \{\pi_i\}_{i\in[\![t]\!]^*})$.

*$\Pi$ is said to be* confidential *if there exists a negligible function* $\epsilon$ *s.t. for any p.p.t. algorithms* $\mathcal{D}$*, any integer* $t$*, any* $\alpha^* \in [\![n]\!]^*$*, any map* $\mathrm{map} \in \mathcal{M}$*, and any tuple of clues* $(\mathrm{clue}_\alpha)_{\alpha\in[\![n]\!]^*} \in \mathcal{C}^n$*, there exists a p.p.t. cryptid simulator* Sim *s.t., considering the following experiment:*

$\mathbf{Exp}^{\mathsf{Confidentiality}}_{\Pi,\mathcal{D},b}(\lambda)$*:*
$\mathrm{set} \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{view}_{\alpha^*} \leftarrow \mathsf{Cryptid}_{\alpha^*,t}\langle\{\mathrm{Player}_\alpha(\mathrm{set},\mathrm{map})\}_{\alpha\in[\![n]\!]^*},$
$\qquad\qquad \mathrm{Master}(\mathrm{set},\mathrm{map},(\mathrm{clue}_\alpha)_{\alpha\in[\![n]\!]^*})\rangle$
*parse* $\mathsf{view}_{\alpha^*}$ *as* $(\mathsf{sc}^0_{\alpha^*}, (\mathsf{pc}^0_\alpha)_{\alpha\in[\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi^0_i)\}_{i\in[\![t]\!]^*})$
$(\mathsf{sc}^1_{\alpha^*}, (\mathsf{pc}^1_\alpha)_{\alpha\in[\![n]\!]^*}, \{\pi^1_i\}_{i\in[\![t]\!]^*}) \leftarrow \mathrm{Sim}(\mathrm{set}, t, \mathrm{map}, \alpha^*, \mathrm{clue}_{\alpha^*}, \{(\alpha_i, A_i, j_i)\}_{i\in[\![t]\!]^*})$
$b' \leftarrow \mathcal{D}(\mathrm{set}, t, \mathrm{map}, \alpha^*, \mathrm{clue}_{\alpha^*}, \mathsf{sc}^b_{\alpha^*}, (\mathsf{pc}^b_\alpha)_{\alpha\in[\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi^b_i)\}_{i\in[\![t]\!]^*})$
*return* $b = b'$

*we have:* $\left|\Pr\left[0 \leftarrow \mathit{Exp}^{\mathit{Confidentiality}}_{\Pi,\mathcal{D},0}(\lambda)\right] - \Pr\left[1 \leftarrow \mathit{Exp}^{\mathit{Confidentiality}}_{\Pi,\mathcal{D},1}(\lambda)\right]\right| \leq \epsilon(\lambda)$.

### 4.2 Our Cryptographic Cryptid

We build two cryptographic cryptid schemes based on two different cryptid encodings. These two schemes provide two different trade-offs between the size of the committed clues and the size/efficiency of the proofs generated during the game. Both schemes are based on the same paradigm: the clues are committed in ElGamal ciphertexts, and the proofs are built from NIP of correct/incorrect decryption of these ciphertexts.

The first encoding we propose, called *map-based clue cryptid encoding*, frees the clue encoding from its actual structure (based on cell types and properties): it gives only the answer the user must give for each of the map cells.

**Definition 10 (MBC Cryptid Encoding).** *The* Map-Based Clue *(MBC) cryptid encoding for* $n$ *players is a tuple* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$ *where* $N = 108$*,* $\mathcal{C} = \{0,1\}^N$*,* $\mathrm{map} = (\mathrm{cell}_i)_{i\in[\![N]\!]^*}$ *where each* $\mathrm{cell}_i$ *describes the properties of the corresponding cell (however this information is encoded), and* Answer *is defined by:*
$\mathsf{Answer}(\mathrm{map}, \mathrm{clue}, j)$**:** *parses* $\mathrm{clue} = (\mathrm{clue}_i)_{i\in[\![N]\!]^*}$ *and returns* $A = \mathrm{clue}_j$.

Based on this encoding, we can commit the clue by encrypting each of the answers for each of the cells. So, when a player answers a question concerning a given cell, they can prove with a NIP that their answer corresponds to the one encrypted in the ciphertext corresponding to the cell in their committed clue.

**Definition 11 (CC1 scheme).** *The cryptographic cryptid scheme* $\mathit{CC1} = (\mathsf{Setup}, \mathsf{GenClue}, \mathsf{OpenClue}, \mathsf{Play}, \mathsf{Verify})$ *for the MBC cryptid encoding* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$ *is defined as follows:*

$\mathsf{Setup}(\lambda)$**:** *generates a group* $\mathbb{G}$ *of prime order* $p$ *generated by* $g$ *s.t.* $p > 2^\lambda$ *and instanciated the ElGamal encryption* $\mathit{ElGamal} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *by* $\mathbb{G}$*, and returns* $\mathrm{set} \leftarrow (\lambda, \mathit{ElGamal})$.

$\mathsf{GenClue}(\text{clue})$**:** *parses* clue *as* $(\text{clue}_i)_{i\in[\![N]\!]^*}$*, generates* $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(\lambda)$*, computes* $E_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\text{clue}_i)$ *for each* $i \in [\![N]\!]^*$*, sets* $E \leftarrow (E_i)_{i\in[\![N]\!]^*}$*,* $\mathsf{pc} \leftarrow (\mathsf{pk}, E)$ *and* $\mathsf{sc} \leftarrow \mathsf{sk}$*, then returns* $(\mathsf{pc},\mathsf{sc})$*.*

$\mathsf{OpenClue}(\mathsf{pc},\mathsf{sc})$**:** *parses* pc *as* $(\mathsf{pk}, (E_i)_{i\in[\![N]\!]^*})$*, checks that* $\mathsf{pk} = g^{\mathsf{sc}}$ *(otherwise aborts), computes* $\text{clue}_i \leftarrow \mathsf{Dec}_{\mathsf{sc}}(E_i)$ *for each* $i \in [\![N]\!]^*$*, then returns* $\text{clue} = (\text{clue}_i)_{i\in[\![N]\!]^*}$*.*

$\mathsf{Play}(\mathsf{pc},\mathsf{sc},\text{map},j,A)$**:** *parses* $(\mathsf{pc},\mathsf{sc})$ *as* $((\mathsf{pk},(E_i)_{i\in[\![N]\!]^*}),\mathsf{sk})$*, computes the proof* $\pi \leftarrow \mathsf{NIP}\,\{\mathsf{sk} : (\mathsf{Dec}_{\mathsf{sk}}(E_j) = A)\}$*, then returns* $\pi$*.*

$\mathsf{Verify}(\pi,\mathsf{pc},\text{map},j,A)$**:** *parses* $(\mathsf{pc},\mathsf{sc})$ *as* $((\mathsf{pk},(E_i)_{i\in[\![N]\!]^*}),\mathsf{sk})$*, verifies the proof* $\pi$ *on the statement* $(\mathsf{pk}, E_j, A)$*, then returns* 1 *if the proof is valid,* 0 *otherwise.*

Theorem 1 and 2 claim that CC1 is sound and confidential.

**Theorem 1.** *If the* NIP *used in CC1 is sound, then CC1 is sound.*

**Theorem 2.** *If the* NIP *used in CC1 is zero-knowledge and ElGamal is IND-CPA, then CC1 is confidential.*

Proofs are given in Appendix B. The computation time of the algorithm Play and the size of the proof $\pi$ are in $O(1)$. In return, the computation time of the algorithm GenClue and the size of each public clue key pc it generates are in $O(N)$, where $N = 108$ in the original cryptid game. As these public clue keys must be produced a priori for each player (between 3 and 5) for each possible game (180 in the original cryptid game) before acquiring the game, it seems worthwhile to reduce the size of these elements.

To this end, we propose another cryptid encoding, called the property-based clue cryptid encoding, which is closer to the clue structure as presented in the original game. Each cell is encoded by its type (desert, forest, *etc.*) and neighborhood properties (one cell from a desert, two cells from a bear's territory, etc.). Clues are divided into two kinds: $(i)$ those indicating that the type of the cell is one of two given types, and $(ii)$ those indicating one of the cell's neighborhood properties. We encode them with a triplet $(C_1, C_2, C_3)$ s.t. if the clue is of kind $(i)$, then $C_1$ and $C_2$ are the two types given by the clue and $C_3$ is undefined, else if the clue is of kind $(ii)$, then $C_3$ is the neighborhood property, and $C_1$ and $C_2$ are undefined.

**Definition 12 (PBC Cryptid Encoding).** *We denote the set containing the* 5 *cell types (desert, forest,* etc.*) by* $\mathcal{T} = \{\bar{T}_l\}_{l\in[\![5]\!]^*}$ *and the set containing the* 14 *neighboring properties (one cell from a desert, two cells from a bear's territory, etc.) by* $\mathcal{P} = \{\bar{P}_l\}_{l\in[\![14]\!]^*}$*. In what follows, for any set* $S$*,* $S_\perp$ *denotes* $S \cup \{\perp\}$*, and* $\mathfrak{P}(S)$ *denotes the power set of* $S$*. The* Property-Based Clue *(PBC) cryptid encoding for* $n$ *players is the tuple* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$ *where* $N = 108$*,* $\mathcal{C} = \mathcal{T}_\perp \times \mathcal{T}_\perp \times \mathcal{P}_\perp$*,* $\mathcal{M}$ *is the set of all the maps* $\text{map} = (\text{cell}_i)_{i\in[\![N]\!]^*}$ *s.t. each* $\text{cell}_i \in \mathcal{T} \times \mathfrak{P}(\mathcal{P})$*, and* Answer *is defined as follows:*

$\mathsf{Answer}(\text{map},\text{clue},j)$**:** *parses each* $\text{cell}_j$ *in* map *as* $(T_j, \mathcal{P}_j)$ *and* clue *as* $(C_1, C_2, C_3)$*. If* $(C_1, C_2) = (\perp, \perp)$*, then if* $C_3 \in \mathcal{P}_j$ *returns* 1*, otherwise* 0*. Else if* $C_3 = \perp$*, then if* $C_1 = T_j$ *or* $C_2 = T_j$ *returns* 1*, otherwise* 0*. Else, aborts.*

We build a scheme based on this encoding where each element of the clue triplet $(C_1, C_2, C_3)$ is committed in a ciphertext. Given a cell on the map, the type and neighborhood properties of that cell, and their answer on that cell, a player can prove the validity of this answer based on their committed clue as follows. If the answer is *yes*, then they show using a $\mathsf{NIP}$ that the type of the cell is one of the two types $C_1$ or $C_2$ encrypted in the committed clue, or that the property $C_3$ encrypted in the committed clue is one of the properties of the cell. Else, if the answer is *no*, then they show using a $\mathsf{NIP}$ that the type of the cell is not one of $C_1$ and $C_2$ encrypted in the committed clue, and that the property $C_3$ encrypted in the committed clue is not one of the properties of the cell.

**Definition 13 (CC2 scheme).** *The cryptographic cryptid scheme* $\mathit{CC2} = (\mathsf{Setup}, \mathsf{GenClue}, \mathsf{OpenClue}, \mathsf{Play}, \mathsf{Verify})$ *for the PBC cryptid encoding* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$ *is defined as follows:*

$\mathsf{Setup}(\lambda)$**:** *generates a group* $\mathbb{G}$ *of prime order* $p$ *generated by* $g$ *s.t.* $p > 2^\lambda$ *and instantiates the ElGamal encryption* $\mathit{ElGamal} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *by* $\mathbb{G}$*. We denote the set containing the* 5 *cell types by* $\mathcal{T}$ *and the set containing the* 14 *neighboring properties by* $\mathcal{P}$*. This algorithm associates a group element picked in the uniform distribution on* $\mathbb{G}$ *with each element in* $\mathcal{T} \cup \mathcal{P} \cup \{\bot\}$ *(it will repeat this process if two different elements are associated with the same group element, which happens with negligible probability* $\leq |\mathcal{T} \cup \mathcal{P} \cup \{\bot\}|^2/p$*). In the following, by abuse of notation, each* $X \in \mathcal{T}$ *(resp.* $X \in \mathcal{P}$ *and* $\bot$*) will denote both the cell type (resp. the neighboring property and the bottom symbol) and the group element associated with it, and so* $\mathcal{T}$ *(resp.* $\mathcal{P}$*) will denote both the cell type set (resp. the neighboring property set) and the set of the group elements associated with all cell types (resp. neighboring properties). It returns* $\mathrm{set} = (\lambda, \mathit{ElGamal}, \mathcal{T}, \mathcal{P}, \bot)$*.*

$\mathsf{GenClue}(\mathrm{clue})$**:** *parses* clue *as* $(C_1, C_2, C_3)$*, generates* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\lambda)$*, computes* $E_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(C_i)$ *for each* $i \in [\![3]\!]^*$*, sets* $\mathsf{pc} \leftarrow (\mathsf{pk}, E_1, E_2, E_3)$ *and* $\mathsf{sc} \leftarrow \mathsf{sk}$*, then returns* $(\mathsf{pc}, \mathsf{sc})$*.*

$\mathsf{OpenClue}(\mathsf{pc}, \mathsf{sc})$**:** *parses* pc *as* $(\mathsf{pk}, E_1, E_2, E_3)$*, checks that* $\mathsf{pk} = g^{\mathsf{sc}}$ *(otherwise aborts), computes* $C_i \leftarrow \mathsf{Dec}_{\mathsf{sc}}(E_i)$ *for each* $i \in [\![3]\!]^*$*, then returns* $\mathrm{clue} = (C_1, C_2, C_3)$*.*

$\mathsf{Play}(\mathsf{pc}, \mathsf{sc}, \mathrm{map}, j, A)$**:** *parses* $(\mathsf{pc}, \mathsf{sc})$ *as* $((\mathsf{pk}, E_1, E_2, E_3), \mathsf{sk})$*,* map *as* $(\mathsf{cell}_i)_{i \in [\![N]\!]^*}$ *and* $\mathsf{cell}_j$ *as* $(T_j, \mathcal{P}_j)$*. If* $A = 1$*, this algorithm computes and returns:*

$$\pi \leftarrow \mathsf{NIP}\left\{\mathsf{sk} : \mathsf{Dec}_{\mathsf{sk}}(E_1) = T_j \vee \mathsf{Dec}_{\mathsf{sk}}(E_2) = T_j \vee \left(\bigvee_{P \in \mathcal{P}_j} \mathsf{Dec}_{\mathsf{sk}}(E_3) = P\right)\right\}.$$

*Else, if* $A = 0$ *it computes and returns:*

$$\pi \leftarrow \mathsf{NIP}\left\{\mathsf{sk} : \mathsf{Dec}_{\mathsf{sk}}(E_1) \neq T_j \wedge \mathsf{Dec}_{\mathsf{sk}}(E_2) \neq T_j \wedge \left(\bigwedge_{P \in \mathcal{P}_j} \mathsf{Dec}_{\mathsf{sk}}(E_3) \neq P\right)\right\}.$$

$\mathsf{Verify}(\pi, \mathsf{pc}, \mathrm{map}, j, A)$**:** *parses* $(\mathsf{pc}, \mathsf{sc})$ *as* $((\mathsf{pk}, E_1, E_2, E_3), \mathsf{sk})$*,* map *as* $(\mathsf{cell}_i)_{i \in [\![N]\!]^*}$*, and* $\mathsf{cell}_j$ *as* $(T_j, \mathcal{P}_j)$*. Depending on* $A$*, this algorithm verifies the proof* $\pi$ *on the statement* $(\mathit{ElGamal}, \mathsf{pk}, E_1, E_2, E_3, T_j, \mathcal{P}_j)$*, it returns* 1 *if the proof is valid,* 0 *otherwise.*

Theorem 3 and 4 claim that CC2 is sound and confidential.

**Theorem 3.** *If the* NIP *used in CC2 are sound, then CC2 is sound.*

**Theorem 4.** *If the* NIP *used in CC2 are zero-knowledge and ElGamal is IND-CPA, then CC2 is confidential.*

Proofs are given in Appendix B. The complexity of the proof size and computation time is linear in the number of neighborhood properties $|\mathcal{P}|$ (we will explain how this proof is instantiated and why its complexity is so later in this section). Thus, the computation time of the algorithm Play and the size of the proof $\pi$ it generates are in $O(|\mathcal{P}|)$, where $|\mathcal{P}| = 14$ in the original cryptid game. In return, the computation time of the GenClue algorithm and the size of each committed clue pc it generates are in $O(1)$. We therefore obtain much smaller commitments, at the cost of proofs requiring moderate computing capacity for the players. Depending on the power of the players' devices and their storage capacity, one can choose the most suitable scheme between CC1 and CC2.

NIP *Instantiation and Implementation.* Let $\mathbb{G}$ be a group of prime order $p$ and let $g \in \mathbb{G}$ be a generator. Let $(g_1, g_2) \in \mathbb{G}^2$ and $x \in \mathbb{Z}_p^*$ be. We set $y_1 = g_1^x$ and $y_2 = g_2^x$. An instantiation of the (interactive) proof of discrete logarithms equality $\mathsf{IP}\{x : y_1 = g_1^x \wedge y_2 = g_2^x\}$ is given in [9], and an instantiation of the (interactive) proof of discrete logarithms inequality $\mathsf{IP}\{x : y_1 = g_1^x \wedge y_2 \neq g_2^x\}$ is given in [7]. These proofs are correct, sound, and zero-knowledge. Moreover, these proofs are *sigma protocols*, which means that these protocols consist of three interactions: the prover sends a commitment, the verifier sends a challenge chosen at random in a given set, and the prover sends a response. Consider now an ElGamal public key $\mathsf{pk} = g^{\mathsf{sk}}$ and an ElGamal ciphertext $c = (c_1, c_2) = (g^r, \mathsf{pk}^r m)$. The relation $\mathsf{Dec}_{\mathsf{sk}}(c) = m$ is equivalent to the relation $(\mathsf{pk} = g^{\mathsf{sk}} \wedge c_2/(c_1^{\mathsf{sk}}) = m)$, which is equivalent to $(\mathsf{pk} = g^{\mathsf{sk}} \wedge (c_2/m) = c_1^{\mathsf{sk}})$. Similarly, the relation $\mathsf{Dec}_{\mathsf{sk}}(c) \neq m$ is equivalent to the relation $(\mathsf{pk} = g^{\mathsf{sk}} \wedge (c_2/m) \neq c_1^{\mathsf{sk}})$. Thus, the proofs $\mathsf{IP}\{\mathsf{sk} : \mathsf{Dec}_{\mathsf{sk}}(c) = m\}$ and $\mathsf{IP}\{\mathsf{sk} : \mathsf{Dec}_{\mathsf{sk}}(c) \neq m\}$ can be instantiated by the discrete logarithms equality proof in [9] and the discrete logarithms equality proof in [7].

Now let us consider two relations $\mathcal{R}_1$ and $\mathcal{R}_2$ and two associated proofs $\mathsf{IP}\{w_1 : (s_1, w_1) \in \mathcal{R}_1\}$ and $\mathsf{IP}\{w_2 : (s_2, w_2) \in \mathcal{R}_2\}$ for some pairs of statement/witness $(s_1, w_1)$ and $(s_2, w_2)$, s.t. these two proofs are sigma protocols and use the same challenge set. The proof $\mathsf{IP}\{(w_1, w_2) : (s_1, w_1) \in \mathcal{R}_1 \wedge (s_2, w_2) \in \mathcal{R}_2\}$ can be easily obtained by using the same challenge for the two relations in the two previous proofs. This method can be generalized for any number of relations, and can therefore be used to prove several correct/incorrect decryptions of ciphertexts (note that if the ciphertexts use the same public key, it also proves that the secret keys used to decrypt the ciphertexts are all the same). On the other hand, the generic transformation given in [11] allows us to build a proof $\mathsf{IP}\{w : (s_1, w) \in \mathcal{R}_1 \vee (s_2, w) \in \mathcal{R}_2\}$ under the same hypothesis (the challenge used in the resulted proof comes from the same challenge set as for $\mathcal{R}_1$ or $\mathcal{R}_2$, and the method can be generalized for any number of relations). Thus, this method can be used to prove one correct/incorrect decryption of ciphertext out of several ciphertext decryptions.

These two methods can be used together to produce proofs for boolean relations of correct/incorrect ciphertext decryptions, and thus instantiate all the proofs used in the schemes CC1 and CC2. Note that the size of these proofs and the computation time increase linearly with the number of correct/incorrect ciphertext decryptions in the relation. Finally, since these proofs are also sigma protocols, they can be turned into non-interactive proofs using the Fiat-Shamir transformation [13].

We implemented our schemes in Rust[1] and tested them on a processor 11th Gen Intel® Core™ i7-1185G7 @ 3.00GHz × 8. In Appendix C, we present an analysis of the performance of each algorithm as a function of the clues and answers used. In each case, we give an average execution time and element size. Each algorithm takes between a few tens and a few hundred milliseconds, which shows that our prototype can be used in practice without any latency problems for players. A full game of Cryptid with the 180 game setups requires about 300 megabytes for CC1 and 10 megabytes for CC2, and can therefore be downloaded to any standard device such as a PC or mobile phone.

## 5 Cryptographic Cryptid with Dishonest Game Master

We extend our model to check that the game designer (called the game master) has prepared the clues and the map correctly (*i.e.*, they are honest). The aim is to allow players to check from the start of the game that their clues are correctly formed and that they converge on at least one cell of the map, which guarantees that the game will end. To do this, we also need to extend our formalism and security properties. We call this new primitive *verifiable cryptographic cryptid*.

### 5.1 Formal Model

We start by giving a new definition of cryptid encoding by adding the algorithm CorrectClue, which decides whether a clue is well formed.

**Definition 14 (Verifiable Cryptid Encoding).** *A* verifiable cryptid encoding *is a tuple* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer}, \mathsf{CorrectClue})$ *where* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$ *is a cryptid encoding and* CorrectClue *is defined as follows:*

$\mathsf{CorrectClue}(\mathrm{map}, \mathrm{clue})$**:** *takes as input a map* $\mathrm{map} \in \mathcal{M}$ *and a clue* $\mathrm{clue} \in \mathcal{C}$*. It returns* `true` *or* `false` *depending on whether the clue is well-formed or not according to the rules (“*`true`*” corresponds to bit* 1 *and “*`false`*” to* 0*).*

For instance, in the original version of the game, CorrectClue will test whether the encoded clue corresponds to a clue of the form “the cell is of this or that type” or of the form “the cell is $n$ cells away from a cell with such-and-such a property”. We now define verifiable cryptographic cryptids. This primitive extends cryptographic cryptids by adding a pair of algorithms to prove/verify

[1] The source code for our implementation is available at `https://anonymous.4open.science/r/cryptidscheme`.

that the game (the committed clues and the map) has been built correctly by the game master.

**Definition 15 (Verifiable Cryptographic Cryptid).** *A* verifiable cryptographic cryptid *for a verifiable cryptid encoding* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer}, \mathsf{CorrectClue})$ *is a tuple of polynomial time algorithms* $(\mathsf{Setup}, \mathsf{GenClue}, \mathsf{OpenClue}, \mathsf{ProveGame}, \mathsf{VerifyGame}, \mathsf{Play}, \mathsf{Verify})$ *where* $(\mathsf{Setup}, \mathsf{GenClue}, \mathsf{OpenClue}, \mathsf{Play}, \mathsf{Verify})$ *is a cryptographic cryptid and* ProveGame *and* VerifyGame *are defined as follows:*

$\mathsf{ProveGame}(j, (\mathsf{pc}_\alpha, \mathsf{sc}_\alpha)_{\alpha \in [\![n]\!]^*}, \mathrm{map})$**:** *takes as input the index* $j$ *of the cryptid habitat, the clue keys* $(\mathsf{pc}_\alpha, \mathsf{sc}_\alpha)_{\alpha \in [\![n]\!]^*}$ *of the players, a map* map*, and returns a proof* $\rho$*.*

$\mathsf{VerifyGame}(\rho, (\mathsf{pc}_\alpha)_{\alpha \in [\![n]\!]^*}, \mathrm{map})$**:** *takes as input a proof* $\rho$*, the public clue keys of the players, the map* map *and returns* $b \in \{\texttt{accept}, \texttt{reject}\}$ *("*`accept`*" corresponds to bit* 1 *and "*`reject`*" to* 0*).*

It is therefore necessary to extend the notion of the cryptid protocol to include a phase where the game master proves to the players via the algorithm ProveGame that the public clue keys have been constructed correctly and that the game ends. This phase takes place between the clue key distribution phase and the start of the game phase.

**Definition 16 (Verifiable Cryptid Protocol).** *Let* $\lambda$ *be a security parameter. The* verifiable cryptid protocol *for* $n$ *players instantiated by a verifiable cryptid encoding* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer}, \mathsf{CorrectClue})$ *and a verifiable cryptographic cryptid* $(\mathsf{Setup}, \mathsf{GenClue}, \mathsf{OpenClue}, \mathsf{ProveGame}, \mathsf{VerifyGame}, \mathsf{Play}, \mathsf{Verify})$ *is defined as the verifiable cryptid protocol except that after that each player* $\mathrm{Player}_\alpha$ *has computed* $\mathrm{clue}_\alpha \leftarrow \mathsf{OpenClue}(\mathsf{pc}_\alpha, \mathsf{sc}_\alpha)$*:*

- Master *runs* $\rho \leftarrow \mathsf{ProveGame}(j, (\mathsf{pc}_\alpha, \mathsf{sc}_\alpha)_{\alpha \in [\![n]\!]^*}, \mathrm{map})$*, then sends* $\rho$ *to each* $\mathrm{Player}_\alpha$ *where* $\alpha \in [\![n]\!]^*$ *and* $j$ *is the index of the cell where the cryptid is.*
- *Each player* $\mathrm{Player}_\alpha$ *checks that* $\mathsf{VerifyGame}(\rho, (\mathsf{pc}_\alpha)_{\alpha \in [\![n]\!]}, \mathrm{map}) = 1$ *(if not,* $\mathrm{Player}_\alpha$ *aborts the protocol).*
- *The other steps are identical to those of the cryptographic cryptid protocol.*

*At the end of the game, each* $\mathrm{Player}_\alpha$ *for* $\alpha \in [\![n]\!]^*$ *returns* $\mathsf{view}_\alpha = (\mathrm{clue}_\alpha, \mathsf{sc}_\alpha, (\mathsf{pc}_{\alpha'})_{\alpha' \in [\![n]\!]^*}, \rho, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i \in [\![t]\!]^*})$*, where* $t$ *is the number of plays.*

A verifiable cryptid protocol must verify the same properties as a cryptid protocol (soundness and confidentiality). These properties do not need to be redefined. In addition, the security model must capture cases where a dishonest game master tries to give players incorrect clues, or clues that do not converge on at least one cell of the map. We call this security property *game soundness*. More specifically, a verifiable cryptid protocol is game-sound if no adversary is able to forge a valid proof, even if some clues are incorrect or there is no cell that matches all the clues. To verify that the adversary has succeeded in its attack, we need to extract the clues from the players' public clue keys. To do this, we need to define an algorithm, not necessarily in polynomial time, to retrieve the clues from the public keys, but without the secret keys. Note that the fact that this algorithm is not polynomial time is not a problem, since it will only be used in the experiment to test the adversary's success, never by a player.

**Definition 17 (Game Soundness).** *Let $\lambda$ be a security parameter and $\Pi = (\mathsf{Setup}, \mathsf{GenClue}, \mathsf{ProveGame}, \mathsf{VerifyGame}, \mathsf{OpenClue}, \mathsf{Play}, \mathsf{Verify})$ be a verifiable cryptid for a verifiable cryptid encoding $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer}, \mathsf{CorrectClue})$. $\Pi$ is said to be* game sound *if there exists a (not necessary p.p.t.) algorithm $\mathsf{OpenClue}^*$ s.t. $\mathsf{OpenClue}(\mathsf{pc}, \mathsf{sc}) = \mathsf{OpenClue}^*(\mathsf{pc})$ for any $(\mathsf{pc}, \mathsf{sc}) \leftarrow \mathsf{GenClue}(\mathrm{clue})$ and the probability that the following experiment returns 1 is negligible in $\lambda$ for any p.p.t. algorithm $\mathcal{A}$.*

$\mathbf{Exp}^{\mathsf{Game\text{-}Soundness}}_{\Pi,\mathcal{A}}(\lambda)$*:*
$\mathrm{set} \leftarrow \mathsf{Setup}(\lambda)$
$(\rho, \mathrm{map}, (\mathsf{pc}_\alpha)_{\alpha \in [\![n]\!]^*}) \leftarrow \mathcal{A}(\mathrm{set})$
$\forall \alpha \in [\![n]\!]^*,\ \mathrm{clue}_\alpha \leftarrow \mathsf{OpenClue}^*(\mathsf{pc}_\alpha)$
*return* $((\mathrm{clue}_\alpha)_{\alpha \in [\![n]\!]^*} \in \mathcal{C}) \wedge (\mathrm{map} \in \mathcal{M})$
$\wedge\ \mathsf{VerifyGame}(\rho, (\mathsf{pc}_\alpha)_{\alpha \in [\![n]\!]^*}, \mathrm{map})$
$\wedge\ ((\exists \alpha \in [\![n]\!]^*\ s.t.\ 0 \leftarrow \mathsf{CorrectClue}(\mathrm{map}, \mathrm{clue}_\alpha))$
$\vee\ (\forall i \in [\![N]\!]^*, \exists \alpha \in [\![n]\!]^*\ s.t.\ 0 \leftarrow \mathsf{Answer}(\mathrm{map}, \mathrm{clue}_\alpha, i)))$

### 5.2 Our Verifiable Cryptographic Cryptid

We are now going to build a verifiable cryptographic cryptid. To do this, we will extend the PBC encoding and the CC2 scheme. Note that the PBC encoding is much more suitable than the MBC encoding for building a verifiable cryptographic cryptid: indeed, the MBC encoding is not based on the actual structure of the clue, but on the answers it gives for each cell, and it is tedious to prove that the structure of the clue is correct just by looking at the answers it gives for each cell on the map. Note also that even if the game master has proven that the game ends, it is also very important for the smooth running of the game to prove that the clues are well constructed, as this could give a player a huge advantage and be very difficult for other players to guess. For example, with MBC encoding in the CC1 scheme, a malicious game master could give a clue to a player that only answers true on the cryptid cell.

**Definition 18 (VPBC Cryptid Encoding).** *By using the same notation as in Definition 12, we define the* Verifiable Property-Based Clue *(VPBC) verifiable cryptid encoding for n players as the tuple $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer}, \mathsf{CorrectClue})$, where $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer})$ is the PBC and $\mathsf{CorrectClue}$ is defined as follows:*

$\mathsf{CorrectClue}(\mathrm{map}, \mathrm{clue})$**:** *takes as input a map $\mathrm{map} \in \mathcal{M}$ and a clue $\mathrm{clue} \in \mathcal{C}$, and parses* clue *as $(C_1, C_2, C_3)$. If $(((C_1, C_2) \in \mathcal{T}^2 \wedge C_1 \neq C_2 \wedge C_3 = \perp) \vee (C_1 = C_2 = \perp \wedge C_3 \in \mathcal{P}))$, then it returns 1 ("`true`"), else it returns 0 ("`false`").*

*Naive Solution to Prove that the Game Ends.* A first idea is to prove that the game ends directly from the public clues and the map. The game ends when there is a cell s.t. each player's clue returns the answer 1 on that cell. We already know how to show that a player's clue returns 1 on a cell (see the Play algorithm of CC2), so we just need to extend this proof to check that a cell in the map verifies this property for each of the clues. Using the methods given in Section 4.2, we

can build the following proof:

$$\mathsf{NIP}\left\{(\mathsf{sk}_\alpha)_{\alpha\in[\![n]\!]^*} : \bigvee_{i=1}^{N}\left(\bigwedge_{\alpha=1}^{n}\left(\begin{array}{c}\mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,1})=T_i \vee \mathsf{Dec}_{\mathsf{sk}}(E_{\alpha,2})=T_i\\ \vee\left(\bigvee_{P\in\mathcal{P}_i}\mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,3})=P\right)\end{array}\right)\right)\right\},$$

where $\mathsf{pc}_\alpha = (\mathsf{pk}_\alpha, E_{\alpha,1}, E_{\alpha,2}, E_{\alpha,3})$ for each $\alpha \in [\![n]\!]^*$ and map $= \{(T_i, \mathcal{P}_i)\}_{i\in[\![N]\!]^*}$. However, the time and size complexity of this proof is $O(nN|\mathcal{P}|)$, which is not very efficient, especially since the full Cryptid game contains 180 games, and so its cryptographic version will contain 180 of such proofs, which must be downloaded when the game is acquired .

*Efficient Game Proof.* We propose a solution where the size complexity of the proof that the game ends and the clues are correct is in $O(N+n(|\mathcal{P}|+|\mathcal{T}|))$. The time complexity is in $O(N|\mathcal{P}|+n(|\mathcal{P}|+|\mathcal{T}|))$, but if we neglect the computation time of a multiplication in $\mathbb{G}$ (a multiplication is much less computationally expensive than an exponentiation in a prime order group), it goes down to $O(N+n(|\mathcal{P}|+|\mathcal{T}|))$. Since $N$ is the largest parameter, this solution is more efficient than the previous one, as $N$ has no factors in the time and size complexity expression.

The main idea is to first commit the cryptid cell. To do this, the game master encrypts the type and each of the neighbourhood properties of the cell in different ciphertexts. More precisely, they encrypt the type in a ciphertext $E_0$, and if the cryptid cell verifies the $i^{\text{th}}$ property in $\mathcal{P}$ then they encrypt this property in $E_i$, otherwise they encrypt the neutral element $1_\mathbb{G}$. The game master constructs a $\mathsf{NIP}$ $\rho_1$ that each $E_i$ correctly encrypts either the $i^{\text{th}}$ property or $1_\mathbb{G}$ (with time/size complexity in $O(|\mathcal{P}|)$). The game master then hashes each cell of the map by computing the product of the group elements corresponding to the type and properties of each cell (this operation requires $N|\mathcal{P}|$ group element multiplications but no exponentiation). This hash function is known to be collision resistant under the discrete logarithm assumption. This result has been proved in [5] and is recalled in the following theorem.

**Theorem 5.** *Let $n$ be an integer, $\lambda$ be a security parameter, and $\mathbb{G}$ be a group of prime order $p$ s.t. $p > 2^\lambda$ and the discrete logarithm assumption holds in $\mathbb{G}$ (i.e., given a random pair $(g,h) \xleftarrow{\$} \mathbb{G}^2$, the probability that any p.p.t. algorithm outputs $\log_g(h)$ is negligible). For any p.p.t. algorithm $\mathcal{A}$, there exists a negligible function $\epsilon_{n\text{-}col}(\lambda)$ s.t.:*
$\Pr\left[g \xleftarrow{\$} \mathbb{G}^n; (x,y) \leftarrow \mathcal{A}(g); \ : \ x,y \in \{0,1\}^n \wedge x \neq y \wedge \prod_{i=0}^{n-1} g_i^{x_i} = \prod_{i=0}^{n-1} g_i^{y_i}\right]$
$\leq \epsilon_{n\text{-}col}(\lambda)$.

Using the ElGamal homomorphism property, any user can compute the encryption of the hash of the cell committed in the $E_i$. The game master can then build a $\mathsf{NIP}$ $\rho_0$ that the hash of the committed cell corresponds to one of the cells on the map (with time/size complexity in $O(N)$). To prove that the game ends, the game master just has to prove in $\rho_{2,\alpha}$ using a $\mathsf{NIP}$ that each player's clue (indexed by $\alpha$) answers 1 on the committed cryptid cell (with time/size

complexity in $O(|\mathcal{P}|)$ for each of the $n$ proofs), in a similar way to the proof of the Play algorithm.

Finally, the game master proves that each of the $n$ clues is well formed by showing that either the first two elements of the clue encrypt two different types in $\mathcal{T}$ and the third element encrypts $\perp$, or that the first two elements encrypt $\perp$ and the last element encrypts one of the properties of $\mathcal{P}$ (with time/size complexity in $O(|\mathcal{T}| + |\mathcal{P}|)$ for each of the $n$ proofs).

**Definition 19 (VCC scheme).** *The verifiable cryptographic cryptid scheme* $\mathit{VCC} = (\mathsf{Setup}, \mathsf{GenClue}, \mathsf{ProveGame}, \mathsf{VerifyGame}, \mathsf{OpenClue}, \mathsf{Play}, \mathsf{Verify})$ *for the VPBC cryptid encoding* $(n, \mathcal{C}, N, \mathcal{M}, \mathsf{Answer}, \mathsf{CorrectClue})$ *is defined as CC2 except that* Setup, ProveGame *and* VerifyGame *are defined as follows:*

$\mathsf{Setup}(\lambda)$**:** *same as in CC2, except that it will repeat the association between the elements of* $\mathcal{T} \cup \mathcal{P} \cup \{\perp\}$ *and the random elements of* $\mathbb{G}$ *process if at least one element is associated with* $1_{\mathbb{G}}$*, which happens with negligible probability* $\leq |\mathcal{T} \cup \mathcal{P} \cup \{\perp\}|/p$.

$\mathsf{ProveGame}(j, (\mathsf{pc}_\alpha, \mathsf{sc}_\alpha)_{\alpha\in[\![n]\!]^*}, \mathrm{map})$**:** *parses* $(\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}$ *as* $(\mathsf{pk}_\alpha, E_{\alpha,1}, E_{\alpha,2}, E_{\alpha,3})_{\alpha\in[\![n]\!]^*}$, $(\mathsf{sc}_\alpha)_{\alpha\in[\![n]\!]^*}$ *as* $(\mathsf{sk}_\alpha)_{\alpha\in[\![n]\!]^*}$, map *as* $(\mathrm{cell}_i)_{i\in[\![N]\!]^*}$ *and each* $\mathrm{cell}_i$ *as* $(T_i, \mathcal{P}_i)$*, generates* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\lambda)$*, computes* $E_0 \leftarrow \mathsf{Enc}_{\mathsf{pk}}(T_j)$*, and for all* $l \in [\![14]\!]^*$*:*

$$E_l = \begin{cases} \mathsf{Enc}_{\mathsf{pk}}(\bar{P}_l) \text{ if } \bar{P}_l \in \mathcal{P}_j \\ \mathsf{Enc}_{\mathsf{pk}}(1_{\mathbb{G}}) \text{ else} \end{cases},$$

*For all* $i \in [\![N]\!]^*$*, it sets* $\mathit{H}(\mathrm{cell}_i) = \prod_{P\in\mathcal{P}_i} P$*. It generates the following* NIP*:*

$$\rho_0 \leftarrow \mathsf{NIP}\left\{\mathsf{sk} : \bigvee_{i\in[\![N]\!]^*} \mathsf{Dec}_{\mathsf{sk}}\left(\prod_{l\in[\![14]\!]^*} E_l\right) = \mathit{H}(\mathrm{cell}_i) \wedge \mathsf{Dec}_{\mathsf{sk}}(E_0) = T_i\right\},$$

$$\rho_1 \leftarrow \mathsf{NIP}\left\{\mathsf{sk} : \bigwedge_{l\in[\![14]\!]^*} \mathsf{Dec}_{\mathsf{sk}}(E_l) = \bar{P}_l \vee \mathsf{Dec}_{\mathsf{sk}}(E_l) = 1_{\mathbb{G}}\right\}.$$ *Then for each* $\alpha \in [\![n]\!]^*$*, it generates the two following* NIP*:*

$$\rho_{2,\alpha} \leftarrow \mathsf{NIP}\left\{\mathsf{sk}, \mathsf{sk}_\alpha : \begin{array}{c} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,1}) = \mathsf{Dec}_{\mathsf{sk}}(E_0) \\ \vee \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,2}) = \mathsf{Dec}_{\mathsf{sk}}(E_0) \\ \vee \left(\bigvee_{l\in[\![14]\!]^*} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,3}) = \mathsf{Dec}_{\mathsf{sk}}(E_l)\right) \end{array}\right\},$$

$$\rho_{3,\alpha} \leftarrow \mathsf{NIP}\left\{\mathsf{sk}_\alpha : \begin{array}{c} \left(\begin{array}{c} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,3}) = \perp \wedge \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,1}) \neq \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,2}) \\ \wedge \left(\bigvee_{\bar{T}\in\mathcal{T}} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,1}) = \bar{T}\right) \\ \wedge \left(\bigvee_{\bar{T}\in\mathcal{T}} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,2}) = \bar{T}\right) \end{array}\right) \\ \vee \left(\begin{array}{c} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,1}) = \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,2}) = \perp \\ \wedge \left(\bigvee_{\bar{P}\in\mathcal{P}} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,3}) = \bar{P}\right) \end{array}\right) \end{array}\right\}.$$

*Finally, it returns* $\rho \leftarrow (\mathsf{pk}, (E_l)_{l\in[\![14]\!]^*}, \rho_0, \rho_1, (\rho_{2,\alpha})_{\alpha\in[\![n]\!]^*}, (\rho_{3,\alpha})_{\alpha\in[\![n]\!]^*})$.

$\mathsf{VerifyGame}(\rho, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \mathrm{map})$**:** *parses* $\rho$ *as* $(\mathsf{pk}, (E_l)_{l\in[\![14]\!]}, \rho_0, \rho_1, (\rho_{2,\alpha})_{\alpha\in[\![n]\!]^*}, (\rho_{3,\alpha})_{\alpha\in[\![n]\!]^*})$, $(\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}$ *as* $(\mathsf{pk}_\alpha, E_{\alpha,1}, E_{\alpha,2}, E_{\alpha,3})_{\alpha\in[\![n]\!]^*}$, map *as* $(\mathrm{cell}_i)_{i\in[\![N]\!]^*}$ *and each* $\mathrm{cell}_i$ *as* $(T_i, \mathcal{P}_i)$*. Verifies the proofs* $\rho_0$ *on the statement* $(\mathsf{ElGamal}, \mathsf{pk}, \prod_{l\in[\![14]\!]^*} E_l, (T_i, H(\mathrm{cell}_i))_{i\in[\![N]\!]^*}, E_0)$*,* $\rho_1$ *on the statement* $(\mathsf{ElGamal}, \mathsf{pk}, (E_l)_{l\in[\![14]\!]^*}, \mathcal{P}, 1_{\mathbb{G}})$*, and for all* $\alpha \in [\![n]\!]^*$*,* $\rho_{2,\alpha}$ *on the statement* $(\mathsf{ElGamal}, \mathsf{pk}_\alpha, E_{\alpha,1}, E_{\alpha,2}, E_{\alpha,3}, \mathsf{pk}, (E_l)_{l\in[\![14]\!]})$ *and* $\rho_{3,\alpha}$ *on the statement* $(\mathsf{ElGamal}, \mathsf{pk}_\alpha, E_{\alpha,1}, E_{\alpha,2}, E_{\alpha,3}, \mathcal{T}, \mathcal{P}, \bot)$*. It returns* 1 *if proofs are valid,* 0 *otherwise.*

The following theorems claim that VCC is sound, game-sound, and confidential.

**Theorem 6.** *If the* NIP *used in VCC are sound, then VCC is sound.*

**Theorem 7.** *If the* NIP *used in VCC are extractable, then VCC is game-sound under the discrete logarithm assumption.*

**Theorem 8.** *If the* NIP *used in VCC are zero-knowledge and ElGamal is IND-CPA, then VCC is confidential.*

Theorem 6 is a direct implication of Theorem 3. The proof of the two other theorems are given in Appendix D.

NIP *Instantiation and implementation.* NIP of VCC can be instantiated using the same tools as described in Section 4.2, except for the proofs that involve the equivalence of two decryptions, *i.e.*, those that contain expressions such as $\mathsf{Dec}_{\mathsf{sk}}(c) = \mathsf{Dec}_{\mathsf{sk}'}(c')$ (the proofs $\rho_{2,\alpha}$). Consider two ElGamal public keys $\mathsf{pk} = g^{\mathsf{sk}}$ and $\mathsf{pk}' = g^{\mathsf{sk}'}$, and two ElGamal ciphertexts $c = (c_1, c_2) = (g^r, \mathsf{pk}^r m)$ and $c' = (c'_1, c'_2) = (g^{r'}, \mathsf{pk}'^{r'} m)$. The relation $\mathsf{Dec}_{\mathsf{sk}}(c) = \mathsf{Dec}_{\mathsf{sk}'}(c')$ is equivalent to the relation $(\mathsf{pk} = g^{\mathsf{sk}} \wedge \mathsf{pk}' = g^{\mathsf{sk}'} \wedge c_2/(c_1^{\mathsf{sk}}) = c'_2/(c_1'^{\mathsf{sk}'}))$, which is equivalent to $(\mathsf{pk} = g^{\mathsf{sk}} \wedge \mathsf{pk}' = g^{\mathsf{sk}'} \wedge (c_2/c'_2) = c_1^{\mathsf{sk}}(c_1'^{-1})^{\mathsf{sk}'})$. By renaming the variables, we obtain the relation $(y_1 = g_1^{x_1} \wedge y_2 = g_2^{x_2} \wedge y = g^{x_1}h^{x_2})$. We give an extractable zero-knowledge sigma-protocol NIP for this relation, which can therefore be used in relations containing "or" and "and" operators as it is the case in $\rho_{2,\alpha}$.

$\mathsf{NIP}\,\{x_1, x_2 : y_1 = g_1^{x_1} \wedge y_2 = g_2^{x_2} \wedge y = g^{x_1}h^{x_2}\}$**:** picks $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p^*$, sets $R_1 \leftarrow g_1^{r_1}$, $R_2 \leftarrow g_2^{r_2}$, $R \leftarrow g^{r_1}$, and $S \leftarrow h^{r_2}$, then hashes $(g_1, g_2, g, h, y_1, y_2, y, R_1, R_2, R, S)$ using a random oracle in order to obtain a challenge $c \in \mathbb{Z}_p^*$, and computes $z_1 \leftarrow r_1 + cx_1$, and $z_2 \leftarrow r_2 + cx_2$. It returns $\pi \leftarrow (R_1, R_2, R, S, z_1, z_2)$.

$\mathsf{Ver}(s, \pi)$**:** parses $s$ as $(g_1, g_2, g, h, y_1, y_2, y, R_1, R_2, R, S)$ and $\pi$ as $(R_1, R_2, R, S, z_1, z_2)$, and verifies that $g_1^{z_1} = R_1 y_1^c$ and $g_2^{z_2} = R_2 y_2^c$ and $g^{z_1}h^{z_2} = RSy^c$.

This NIP is extractable (so sound). Indeed, we can build the following extractor: runs the prover and simulates the random oracle to the prover in order to obtain a valid transcript $(R_1, R_2, R, S, z_1, z_2)$ for a challenge $c$, then rewinds the prover to the challenge step in order to obtain a second valid transcript $(R_1, R_2, R, S, z'_1, z'_2)$ having the same commitments $(R_1, R_2, R, S)$ for another challenge $c'$ outputted by the random oracle (tries again if the prover fails). The witness $(x_1, x_2)$ can be efficiently extracted from these values by computing $(x_1, x_2) = ((z_1 - z'_1)/(c - c')^{-1}, (z_2 - z'_2)(c - c')^{-1})$.

This NIP is also zero-knowledge. Indeed, we can build the following simulator: picks $c, z_1, z_2 \xleftarrow{\$} \mathbb{Z}_p^*$ and $S \xleftarrow{\$} \mathbb{G}$, and computes $R_1 = g_1^{z_1} y_1^{-c}$ and $R_2 = g_2^{z_2} y_2^{-c}$ and $R = g^{z_1} h^{z_2} S^{-1} y^{-c}$. Set $c$ as the hash of $(g_1, g_2, g, h, y_1, y_2, y, R_1, R_2, R, S)$ and returns the transcript $(R_1, R_2, R, S, z_1, z_2)$.

We extended our implemetation of CC2 to make it verifiable, as described in this section. In Appendix E, we present an analysis of the performance of the algorithms ProveGame and VerifyGame as a function of the number of players. The proof that the game is correct takes less than 5 seconds and must be done for the 180 game setups by the game editor. It will therefore take 15 minutes for the whole Cryptid game, bearing in mind that this operation is only performed once, a priori, before the game distribution by the game editor.

The verification of a game takes less than 5 seconds. This operation must be performed by each player (we remind the reader that it is only done once before the start of the game). The size of the proof of each game setup is less than 1.3 megabytes, knowing that these proofs must be downloaded at the same time as the public clue keys when the game is acquired. Downloading the full game will therefore require less than $10 + 180 \times 1.3 = 244$ megabytes of storage.

## 6 Conclusion

We have proposed cryptographic protocols to prevent cheating in the game Cryptid. We have formally defined and proved two security properties for these protocols: soundness and confidentiality. We give an implementation to show that they can be used in a practical context. We have also extended one of these protocols to allow the game designer to prove that the clues are well formed and converge on at least one cell of the map.

However, the cryptid game has a feature that we did not consider. If the players get impatient and all agree, they can consult a booklet to get an extra clue for their game. This clue does not have the same structure as the player's clues; for instance, the general clue might be "there is no clue indicating that the type of the cryptid cell is one of two given types". It would be interesting to add a system for committing these clues, so that if the players agree, they can open these commitments. It would then be necessary for the game designer to prove that these hidden clues are also well formed and consistent with the other clues and the map. We leave the design of a protocol that takes these general clues and the difficult mode (described in Section 2) into account as future works.

## References

1. Barnett, A., Smart, N.P.: Mental Poker Revisited. In: Cryptography and Coding (2003)
2. Bella, R., Bultel, X., Chevalier, C., Lafourcade, P., Olivier-Anclin, C.: Practical Construction for Secure Trick-Taking Games Even With Cards Set Aside. In: FC2023-Twenty-Seventh International Conference Financial Cryptography and Data Security (2023)

3. Blum, M., Feldman, P., Micali, S.: Non-Interactive Zero-Knowledge and Its Applications. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. STOC '88, Association for Computing Machinery (1988)
4. Boneh, D.: The Decision Diffie-Hellman problem. In: International Algorithmic Number Theory Symposium (1998)
5. Brands, S.A.: An Efficient Off-line Electronic Cash System Based On The Representation Problem. (1993)
6. Bultel, X., Lafourcade, P.: Secure Trick-Taking Game Protocols: How to Play Online Spades with Cheaters. In: Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23 (2019)
7. Camenisch, J., Shoup, V.: Practical Verifiable Encryption and Decryption of Discrete Logarithms. In: Advances in Cryptology - CRYPTO 2003 (2003)
8. Canard, S., Pointcheval, D., Sanders, O., Traoré, J.: Divisible e-cash made practical. In: Public-Key Cryptography – PKC 2015 (2015)
9. Chaum, D., Pedersen, T.P.: Wallet Databases with Observers. In: Advances in Cryptology — CRYPTO' 92. Springer (1993)
10. Cortier, V., Gaudry, P., Glondu, S.: Belenios: A Simple Private and Verifiable Electronic Voting System (2019)
11. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: CRYPTO '94 (1994)
12. Evans, D., Kolesnikov, V., Rosulek, M.: A Pragmatic Introduction to Secure Multi-Party Computation. Foundations and Trends in Privacy and Security (2018)
13. Fiat, A., Shamir, A.: How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In: CRYPTO' 86 (1987)
14. Fischlin, M., Mittelbach, A.: An Overview of the Hybrid Argument. Cryptology ePrint Archive, Paper 2021/088 (2021)
15. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In: 25th USENIX Security Symposium (2016)
16. Golle, P.: Dealing Cards in Poker Games. In: International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II (2005)
17. Lindell, Y.: How to Simulate It – A Tutorial on the Simulation Proof Technique (2017)
18. Lovecruft, I.A., de Valence, H.: curve25519_dalek
19. Schindelhauer, C.: A toolbox for mental card games (1998)
20. Shamir, A., Rivest, R.L., Adleman, L.M.: Mental Poker (1981)
21. Shoup, V.: Sequences of Games: A Tool for Taming Complexity in Security Proofs. Cryptology ePrint Archive, Paper 2004/332 (2004)
22. Stamer, H.: Efficient Electronic Gambling: An Extended Implementation of the Toolbox for Mental Card Games. In: Western European Workshop on Research in Cryptology (2005)

## A Full Version of the Rules


**Fig. 2.** Cryptid materials.

Cryptid is a board game designed in 2018 by Hal Duncan and Ruth Veevers, and published by Osprey Games, for 3, 4 or 5 players aged 10 and over. Cryptid is a unique deduction game of honest misdirection in which players must try to uncover information about their opponents' clues while throwing them off the scent of their own. Each player has a clue to help them find the cryptid, and in their turn they can try to get more information from their opponents. Be warned, give away too much and your opponents may find the cryptid faster than you.

The game includes a modular board, five clue books, and a deck of set-up cards with hundreds of possible set-ups across two difficulty levels, as it is shown in Figure 2. More precisely, the game contains:

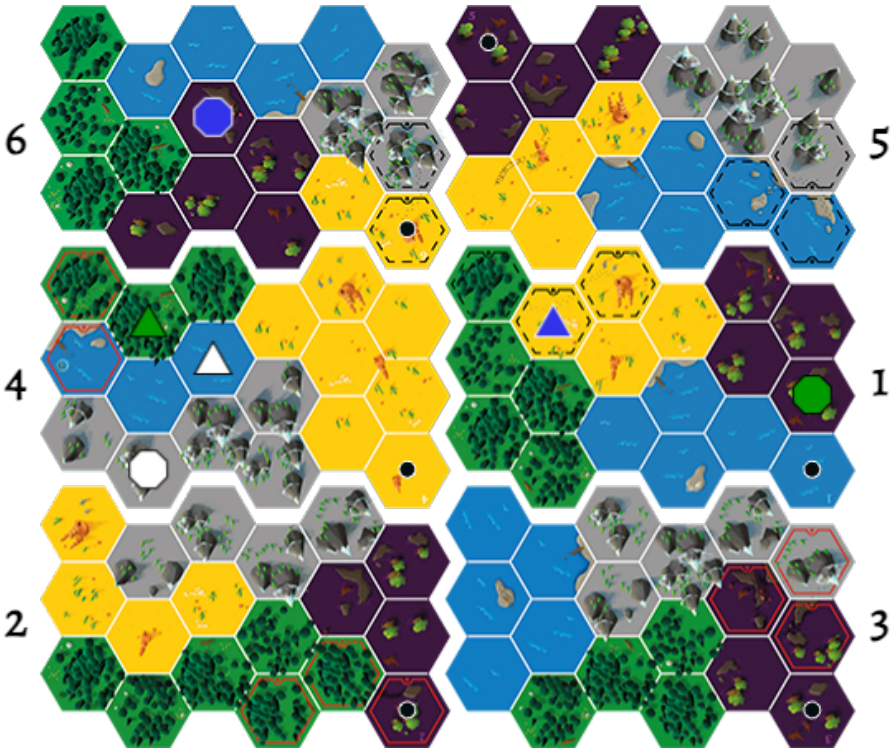
- 60 cards,
- 4 standing Stones (octagon),
- 4 abandoned Shacks (triangles),
- 1 pawn,

– 5 sets of Player Tokens (cubes and discs in matching colours),
– 5 clue Books ($\alpha, \beta, \delta, \gamma, \epsilon$),
– 6 tiles for the map.

### A.1 Setup of the game

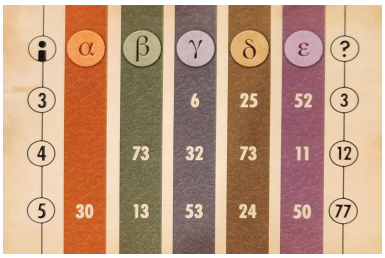
The first step is to choose a card from the 60 cards in the game. On one side of the card is a description of the map, an example is shown in Figure 3. The map of a game is made up of 6 numbered tiles, which are assembled to form a 2 by 3 map. The map is made up of hexagonal cells representing terrains, which can be of 5 types: water (blue), forest (green), desert (yellow), mountain (grey) and swamp (brown). Depending on the map, some blue, green, white or black structures (standing stones are represented by octagons and abandoned huts by triangles) must also be placed on the map. The map may also show some animal territories: dotted lines around cells indicate bear territories and red lines around cells indicate puma territories.

**Fig. 3.** Cryptid example of initial board, represented on one side of the card.



The other side of the card indicates each player a book and a clue number, and each player secretly looks up their clue in the book on the line corresponding to their number. An example is given in Figure 4, where, for a game with 4 players, the clue of the player with the book $\beta$ is on the line 73.

**Fig. 4.** Example of a card of Cryptid. The fist number indicates the number of player for the map. The others indicate the lines of the clue for each clue book.




For example, if we consider 4 player with the map on Figure 3 and the card on Figure 4, the clues that players will read in the books are:

- Player 1: The cryptid habitat is 3 cells away from a green structure
- Player 2: The cryptid habitat is 3 cells away from a white structure
- Player 3: The cryptid habitat is on forest or desert
- Player 4: The cryptid habitat is on water or forest

The list of all possible clues, called deduction sheet, is given to the player (see Figure 6). The game offers two levels of difficulty: the standard mode where the clues are those on the left of the deduction sheet, and the difficult mode where we also consider the negation of all the standard clues (on the right of the deduction sheet). Note that in this work, we only consider the standard version of the game.

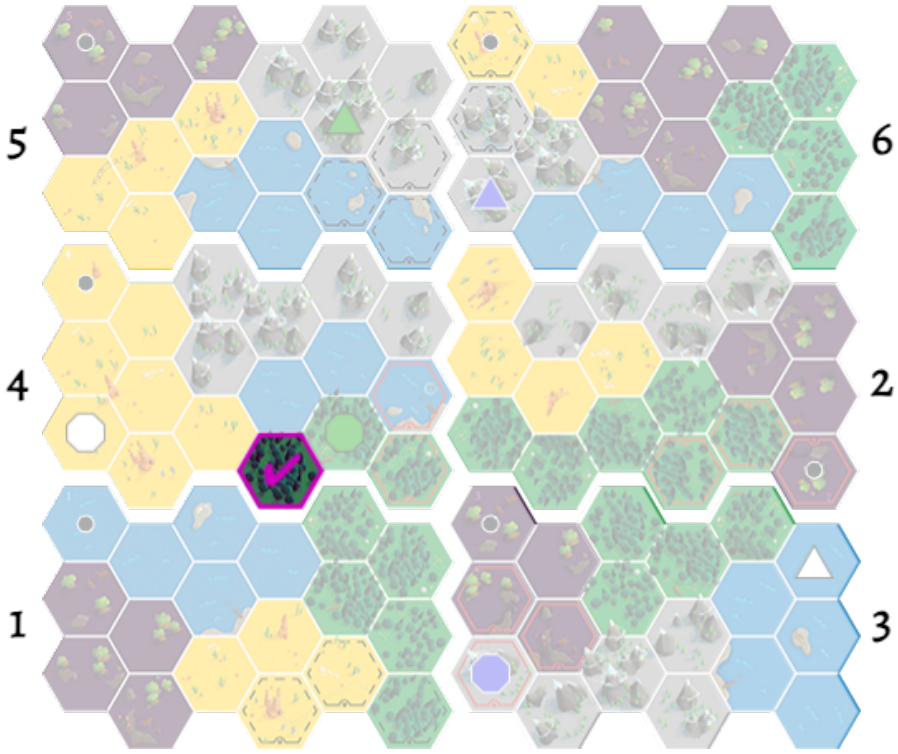
The unique cell that satifies all the clues is shown on Figure 5. To win the game, a player has to find this cell.

### A.2 A turn

When it is their turn, a player has two possible actions: Ask a question or make a search.

*Ask a question.* The player puts the black pawn on a cell with no cubes to asks a chosen player if the cryptid can live there. If the player knows from their clue that the cryptid can live there, they must put a disc on that cell, otherwise the player must put a cube on that cell.

**Fig. 5.** Cryptid solution of the example in Figure 3.



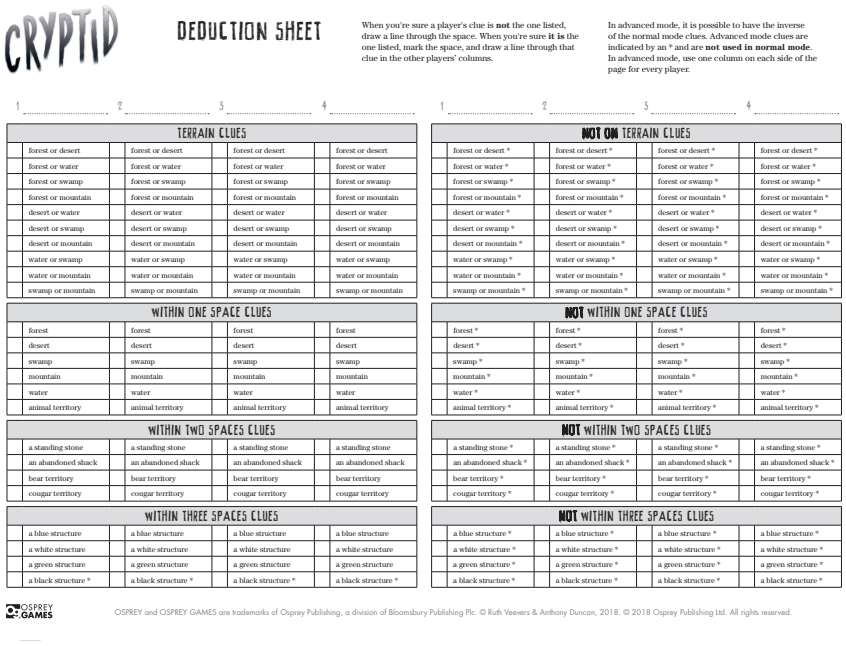

*Search.* The player chooses a cell that their clue indicates could be the cell of the cryptid, and that does not contain a cube. The player puts a disc on the cell, then the other players in turn put a disc if the cell can be the habitat of the cryptid according to their clue, otherwise a cube. As soon as a player places a cube, the search stops. If every player puts a disc, the game ends and the player wins.

*End of the turn.* At the end of your turn, if one of your opponents placed a cube on the map, you must put a cube on a cell of your choice where, according to your clue, the cryptid cannot live.

### A.3 End of the game

The game ends when one player finds the cryptid cell. A free electronic version of the game is available online at `https://www.playcryptid.com/`.

**Fig. 6.** Cryptid possible clues.



CRYPTID

DEDUCTION SHEET

When you're sure a player's clue is **not** the one listed, draw a line through the space. When you're sure **it is** the one listed, mark the space, and draw a line through that clue in the other players' columns.

In advanced mode, it is possible to have the inverse of the normal mode clues. Advanced mode clues are indicated by an * and are **not used in normal mode**. In advanced mode, use one column on each side of the page for every player.

1 2 3 4

| TERRAIN CLUES | | | |
|---|---|---|---|
| forest or desert | forest or desert | forest or desert | forest or desert |
| forest or water | forest or water | forest or water | forest or water |
| forest or swamp | forest or swamp | forest or swamp | forest or swamp |
| forest or mountain | forest or mountain | forest or mountain | forest or mountain |
| desert or water | desert or water | desert or water | desert or water |
| desert or swamp | desert or swamp | desert or swamp | desert or swamp |
| desert or mountain | desert or mountain | desert or mountain | desert or mountain |
| water or swamp | water or swamp | water or swamp | water or swamp |
| water or mountain | water or mountain | water or mountain | water or mountain |
| swamp or mountain | swamp or mountain | swamp or mountain | swamp or mountain |

| WITHIN ONE SPACE CLUES | | | |
|---|---|---|---|
| forest | forest | forest | forest |
| desert | desert | desert | desert |
| swamp | swamp | swamp | swamp |
| mountain | mountain | mountain | mountain |
| water | water | water | water |
| animal territory | animal territory | animal territory | animal territory |

| WITHIN TWO SPACES CLUES | | | |
|---|---|---|---|
| a standing stone | a standing stone | a standing stone | a standing stone |
| an abandoned shack | an abandoned shack | an abandoned shack | an abandoned shack |
| bear territory | bear territory | bear territory | bear territory |
| cougar territory | cougar territory | cougar territory | cougar territory |

| WITHIN THREE SPACES CLUES | | | |
|---|---|---|---|
| a blue structure | a blue structure | a blue structure | a blue structure |
| a white structure | a white structure | a white structure | a white structure |
| a green structure | a green structure | a green structure | a green structure |
| a black structure * | a black structure * | a black structure * | a black structure * |

1 2 3 4

| NOT ON TERRAIN CLUES | | | |
|---|---|---|---|
| forest or desert * | forest or desert * | forest or desert * | forest or desert * |
| forest or water * | forest or water * | forest or water * | forest or water * |
| forest or swamp * | forest or swamp * | forest or swamp * | forest or swamp * |
| forest or mountain * | forest or mountain * | forest or mountain * | forest or mountain * |
| desert or water * | desert or water * | desert or water * | desert or water * |
| desert or swamp * | desert or swamp * | desert or swamp * | desert or swamp * |
| desert or mountain * | desert or mountain * | desert or mountain * | desert or mountain * |
| water or swamp * | water or swamp * | water or swamp * | water or swamp * |
| water or mountain * | water or mountain * | water or mountain * | water or mountain * |
| swamp or mountain * | swamp or mountain * | swamp or mountain * | swamp or mountain * |

| NOT WITHIN ONE SPACE CLUES | | | |
|---|---|---|---|
| forest * | forest * | forest * | forest * |
| desert * | desert * | desert * | desert * |
| swamp * | swamp * | swamp * | swamp * |
| mountain * | mountain * | mountain * | mountain * |
| water * | water * | water * | water * |
| animal territory * | animal territory * | animal territory * | animal territory * |

| NOT WITHIN TWO SPACES CLUES | | | |
|---|---|---|---|
| a standing stone * | a standing stone * | a standing stone * | a standing stone * |
| an abandoned shack * | an abandoned shack * | an abandoned shack * | an abandoned shack * |
| bear territory * | bear territory * | bear territory * | bear territory * |
| cougar territory * | cougar territory * | cougar territory * | cougar territory * |

| NOT WITHIN THREE SPACES CLUES | | | |
|---|---|---|---|
| a blue structure * | a blue structure * | a blue structure * | a blue structure * |
| a white structure * | a white structure * | a white structure * | a white structure * |
| a green structure * | a green structure * | a green structure * | a green structure * |
| a black structure * | a black structure * | a black structure * | a black structure * |

OSPREY GAMES



# B Security Proofs of CC1 and CC2

In this section, we give the security proofs of our schemes.

## B.1 Proof of Theorem 1

The languages $\mathcal{L}$ of the $\mathsf{NIP}$ used in the algorithm $\mathsf{Play}$ of $\mathsf{CC1}$ verifies $s \in \mathcal{L} \Leftrightarrow (\exists\ w, (s,w) \in \mathcal{R})$ where $(s,w) = ((\mathsf{ElGamal}, \mathsf{pk}, E, A), \mathsf{sk})$ and $(s,w) \in \mathcal{R} \Leftrightarrow \mathsf{Dec}_{\mathsf{sk}}(E) = A$.

Let $\lambda$ be a security parameter. We show that if $\mathsf{CC1}$ is not sound, then the $\mathsf{NIP}$ for one of the two languages $\mathcal{L}$ is not sound either. Assume that there exists a pair of p.p.t. (in $\lambda$) algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ s.t. $\epsilon(\lambda) = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Soundness}}_{\mathsf{CC1},\mathcal{A}}(\lambda)\right]$ is non-negligible. We build a p.p.t. algorithm $\mathcal{B}$ that tries to break the $\mathsf{Soundness}$ of a $\mathsf{NIP}$ for $\mathcal{L}$.

$\mathcal{B}(\mathcal{L})$**:** runs $\mathsf{set} \leftarrow \mathsf{Setup}(\lambda)$ and parses $\mathsf{set}$ as $(\lambda, \mathsf{ElGamal})$. It runs $\mathsf{clue} \leftarrow \mathcal{A}_2(\mathsf{set})$, $(\mathsf{pc}, \mathsf{sc}) \leftarrow \mathsf{GenClue}(\mathsf{clue})$, and $(\pi, \mathrm{map}, j, A) \leftarrow \mathcal{A}_2(\mathsf{pc}, \mathsf{sc})$. It parses $(\mathsf{pc}, \mathsf{sc})$ as $((\mathsf{pk}, (E_i)_{i \in [\![N]\!]^*}), \mathsf{sk})$ and returns $((\mathsf{ElGamal}, \mathsf{pk}, E_j, A), \pi)$.

If $\mathcal{A}$ wins its soundness experiment, then $\pi$ is a valid proof. We show that however $(\mathsf{ElGamal}, \mathsf{pk}, E_j, A) \notin \mathcal{L}$. Since $(\mathsf{pc}, \mathsf{sc}) \leftarrow \mathsf{GenClue}(\mathsf{clue})$, we have that $E_j \leftarrow$

$\mathsf{Enc}_{\mathsf{pk}}(\text{clue}_j)$, and since $\mathsf{Answer}(\text{map}, \text{clue}, j) = 1 - A$ and $\mathsf{Answer}(\text{map}, \text{clue}, j) = \text{clue}_j$, we have that $\text{clue}_j = 1 - A \neq A$, so $\mathsf{Dec}_{\mathsf{sk}}(E_j) \neq A$, which implies that $(\mathsf{ElGamal}, \mathsf{pk}, E_j, A) \notin \mathcal{L}$. Thus, $\mathcal{B}$ wins its soundness experiment with the same probability $\epsilon(\lambda)$ as $\mathcal{A}$ wins its own, which is non-negligible.

### B.2 Proof of Theorem 2

Throughout this proof, we will refer to $(\mathbb{G}, g, p)$ as the group $\mathbb{G}$ of prime order $p$ generated by $g$ used in $\mathsf{ElGamal}$. $\mathsf{CC2}$ uses a $\mathsf{NIP}$ in the algorithm $\mathsf{Play}$. Since this proof is assumed to be zero-knowledge, there exists a simulator $\text{Sim}_{\mathsf{NIP}}$ that simulates valid proofs from any statment. We build the following cryptid simulator.

$\text{Sim}(\text{set}, t, \text{map}, \alpha^*, \text{clue}_{\alpha^*}, \{(\alpha_i, A_i, j_i)\}_{i \in [\![t]\!]^*})$**:** runs $(\mathsf{pc}_{\alpha^*}, \mathsf{sc}_{\alpha^*}) \leftarrow \mathsf{GenClue}(\text{set}, \text{clue}_{\alpha^*})$ and parses $\mathsf{pc}_{\alpha^*}$ as $(\mathsf{pk}_{\alpha^*}, (E_{\alpha^*,i})_{i \in [\![N]\!]^*})$. For each $\alpha \in [\![n]\!]^* \backslash \{\alpha^*\}$, picks $\mathsf{pk}_\alpha \xleftarrow{\$} \mathbb{G}$ and $(\gamma_{\alpha,i})_{i \in [\![N]\!]^*} \xleftarrow{\$} \mathbb{G}^N$, for all $i \in [\![N]\!]^*$ computes $E_{\alpha,i} \leftarrow \mathsf{Enc}_{\mathsf{pk}_\alpha}(\gamma_{\alpha,i})$, then sets $\mathsf{pc}_\alpha \leftarrow (\mathsf{pk}_\alpha, (E_{\alpha,i})_{i \in [\![N]\!]^*})$. According to the rules of Cryptid, the simulator simulates each play $i$ for player $\text{Player}_{\alpha_i}$ (from play 1 to play $t$) by computing $\pi_i \leftarrow \text{Sim}_{\mathsf{NIP}}(\mathsf{ElGamal}, \mathsf{pk}_{\alpha_i}, E_{\alpha_i,j_i}, A_i)$. Finally, it returns $\mathsf{view}_{\alpha^*} = (\text{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha \in [\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i \in [\![t]\!]^*})$.

Note that this simulator replaces each ciphertext with the encryption of a random element and simulates each $\mathsf{NIP}$ .

To prove this theorem, we use the following sequence of games [21] where the first game matches the case where the distinguisher receives elements generated by the real protocol (case $b = 0$ in the confidentiality experiment), and where the last game matches the case where the distinguisher receives elements generated by the simulator (case $b = 1$ in the confidentiality experiment). By showing that each of the intermediate games are computationally indistinguishable from the other, we deduce that the two cases are indistinguishable.

**Game** $G_0$**:** $\mathcal{D}$ receives as input $(\text{set}, t, \text{map}, \alpha^*, \text{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha \in [\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i \in [\![t]\!]^*})$ as in the case $b = 0$ (corresponding to the real protocol) in the confidentiality experiment, we have:

$$\Pr[\mathcal{D} \text{ returns } 1 \text{ in } G_0] = \Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{CC1},\mathcal{D},0}(\lambda)\right].$$

**Game** $G_1$**:** This game is the same as $G_0$ except that each proof $\pi_i$ is replaced by a simulated proof. Since all $\mathsf{NIP}$ are zero-knowledge, we have:

$$\Pr[\mathcal{D} \text{ returns } 1 \text{ in } G_0] = \Pr[\mathcal{D} \text{ returns } 1 \text{ in } G_1\ ].$$

At this step, the only difference between $G_1$ and the case $b = 1$ in the confidentiality experiment (corresponding to the simulator) is that in the simulator, ciphertexts are generated from random messages.

**Game $G_2$:** $\mathcal{D}$ receives as input $(\text{set}, t, \text{map}, \alpha^*, \text{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i\in[\![t]\!]^*})$ as in the case $b = 1$ (corresponding to the simulator) in the confidentiality experiment, we have:

$$\Pr[\mathcal{D} \text{ returns 1 in } G_2] = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{CC1},\mathcal{D},1}(\lambda)\right].$$

To prove indistinguishability between $G_1$ and $G_2$, we use an hybrid argument [14]. We consider a sequence of games from $G_1$ to $G_2$ in which we progressively replace each ciphertext in $G_1$ with the encryption of a random element.

**Game $G_{1,\alpha}$ (for all $\alpha \in [\![n]\!]^*$):** We define $G_{1,0}$ as $G_1$, and $G_{1,n}$ as $G_2$. The game $G_{1,\alpha}$ is the same as $G_{1,\alpha-1}$ except that if $\alpha \neq \alpha^*$, then the challenger picks $\mathsf{pk}_\alpha \xleftarrow{\$} \mathbb{G}$ and $(\gamma_{\alpha,i})_{i\in[\![N]\!]^*} \xleftarrow{\$} \mathbb{G}^N$, computes $E_{\alpha,i} \leftarrow \mathsf{Enc}_{\mathsf{pk}_\alpha}(\gamma_{\alpha,i})$ for all $i \in [\![N]\!]^*$ , then sets $\mathsf{pc}_\alpha \leftarrow (\mathsf{pk}_\alpha, (E_{\alpha,i})_{i\in[\![N]\!]^*})$.

**Game $G_{1,\alpha,l}$ (for all $l \in [\![N]\!]^*$):** We define $G_{1,\alpha,0}$ as $G_{1,\alpha-1}$ and $G_{1,\alpha,N}$ as $G_{1,\alpha}$. The game $G_{1,\alpha,l}$ is the same as $G_{1,\alpha,l-1}$ except that if $\alpha \neq \alpha^*$, then it computes $\gamma_{\alpha,l} \xleftarrow{\$} \mathbb{G}$ and sets $E_{\alpha,l} \leftarrow \mathsf{Enc}_{\mathsf{pk}_\alpha}(\gamma_{\alpha,l})$.

We emphasize that $G_{1,\alpha^*,l-1} = G_{1,\alpha^*,l}$. Let $\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda)$ be the IND-CPA advantage of the encryption scheme ElGamal. We claim that, for any $\alpha \in [\![n]\!]^*\backslash\{\alpha^*\}$ and $l \in [\![N]\!]^*$:

$$\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) \geq \left|\Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l-1}] - \Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l}]\right|.$$

We prove this claim by reduction. We build the following two-party p.p.t. algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the IND-CPA experiment on ElGamal:

$\mathcal{A}_1(\mathsf{pk})$**:** sets $\mathsf{pk}_\alpha \leftarrow \mathsf{pk}$, parses $\text{clue}_\alpha$ as $(\text{clue}_{\alpha,i})_{i\in[\![N]\!]^*}$, sets $m_0 = \text{clue}_{\alpha,l}$ and picks $m_1 \xleftarrow{\$} \mathbb{G}$, and returns $(m_0, m_1)$.

$\mathcal{A}_2(c)$**:** sets $E_{\alpha,l} = c$, generates a tuple $\mathsf{input} = (\text{set}, t, \text{map}, \alpha^*, \text{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i\in[\![t]\!]^*})$ as in $G_{1,\alpha,l-1}$ except that it uses its own values $\mathsf{pk}_\alpha$ and $E_{\alpha,l}$ in $\mathsf{pc}_\alpha$, and runs $b' \leftarrow \mathcal{D}(\mathsf{input})$. It returns $b'$.

We have:

$$\Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},0}(\lambda)\right] = \Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l-1}],$$

$$\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},1}(\lambda)\right] = \Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l}].$$

Thus, we deduce:

$$\begin{aligned}\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) &\geq \left|\Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},0}(\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},1}(\lambda)\right]\right| \\ &= \left|\Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l-1}] - \Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l}]\right|,\end{aligned}$$

which concludes the proof of the claim. We deduce that:

$$N(n-1)\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) \geq \left|\Pr[\mathcal{D} \text{ returns 1 in } G_1] - \Pr[\mathcal{D} \text{ returns 1 in } G_2]\right|.$$

Finally:

$$N(n-1)\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) \geq |\Pr[\mathcal{D} \text{ returns 1 in } G_0] - \Pr[\mathcal{D} \text{ returns 1 in } G_2]|$$
$$= \left|\Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{CC1},\mathcal{A},0}(\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{CC1},\mathcal{A},1}(\lambda)\right]\right|,$$

which concludes the proof of the theorem, since $N(n-1)\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda)$ is negligible.

### B.3 Proof of Theorem 3

The languages of the NIP used in the algorithm Play of CC2 depend on the answer $A \in \{0,1\}$. We refer to them respectively as $\mathcal{L}_0$ (when $A = 0$) and $\mathcal{L}_1$ (when $A = 1$). $\mathcal{L}_A$ verifies $s \in \mathcal{L}_A \Leftrightarrow (\exists\ w, (s,w) \in \mathcal{R}_A)$ where $(s,w) = ((\mathsf{ElGamal}, \mathsf{pk}, E_1, E_2, E_3, T_j, \mathcal{P}_j), \mathsf{sk})$ and:

$$(s,w) \in \mathcal{R}_1 \Leftrightarrow \mathsf{Dec}_{\mathsf{sk}}(E_1) = T_j \ \vee\ \mathsf{Dec}_{\mathsf{sk}}(E_2) = T_j \ \vee\ \left(\bigvee_{P \in \mathcal{P}_j} \mathsf{Dec}_{\mathsf{sk}}(E_3) = P\right).$$
$$(s,w) \in \mathcal{R}_0 \Leftrightarrow \mathsf{Dec}_{\mathsf{sk}}(E_1) \neq T_j \ \wedge\ \mathsf{Dec}_{\mathsf{sk}}(E_2) \neq T_j \ \wedge\ \left(\bigwedge_{P \in \mathcal{P}_j} \mathsf{Dec}_{\mathsf{sk}}(E_3) \neq P\right).$$

Let $\lambda$ be a security parameter. We show that if CC2 is not sound, then the NIP for one of the two languages $\mathcal{L}_1$ or $\mathcal{L}_0$ is not sound either. Assume that there exists a pair of p.p.t. (in $\lambda$) algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ s.t. $\epsilon(\lambda) = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Soundness}}_{\mathsf{CC2},\mathcal{A}}(\lambda)\right]$ is non-negligible. We build a p.p.t. algorithm $\mathcal{B}_b$ that tries to break the Soundness of a NIP for $\mathcal{L}_b$.

$\mathcal{B}_b(\mathcal{L}_b)$**:** runs set $\leftarrow \mathsf{Setup}(\lambda)$ and parses set as $(\lambda, \mathsf{ElGamal}, \mathcal{T}, \mathcal{P}, \perp)$. It runs clue $\leftarrow \mathcal{A}_2(\text{set})$, $(\mathsf{pc}, \mathsf{sc}) \leftarrow \mathsf{GenClue}(\text{clue})$, and $(\pi, \text{map}, j, A) \leftarrow \mathcal{A}_2(\mathsf{pc}, \mathsf{sc})$. If $A \neq b$, $\mathcal{B}_b$ aborts, else it parses $(\mathsf{pc}, \mathsf{sc})$ as $((\mathsf{pk}, E_1, E_2, E_3), \mathsf{sk})$, map as $(\mathsf{cell}_i)_{i \in [\![N]\!]^*}$, and $\mathsf{cell}_j$ as $(T_j, \mathcal{P}_j)$, and returns $((\mathsf{ElGamal}, \mathsf{pk}, E_1, E_2, E_3, T_j, \mathcal{P}_j), \pi)$.

If $\mathcal{A}$ wins its soundness experiment, then $\pi$ is a valid proof. We show that however $(\mathsf{ElGamal}, \mathsf{pk}, E_1, E_2, E_3, T_j, \mathcal{P}_j) \notin \mathcal{L}_A$:

- If $A = 1$ then $\mathsf{Answer}(\text{map}, \text{clue}, j) = 1 - A = 0$, so $C_1 \neq T_j$ and $C_2 \neq T_j$ and $C_3 \notin \mathcal{P}_j$, which implies that $(\mathsf{ElGamal}, \mathsf{pk}, E_1, E_2, E_3, T_j, \mathcal{P}_j) \notin \mathcal{L}_1$. Thus, $\mathcal{B}_1$ wins its soundness experiment.
- If $A = 0$ then $\mathsf{Answer}(\text{map}, \text{clue}, j) = 1 - A = 1$, so $C_1 = T_j$ or $C_2 = T_j$ or $C_3 \in \mathcal{P}_j$, which implies that $(\mathsf{ElGamal}, \mathsf{pk}, E_1, E_2, E_3, T_j, \mathcal{P}_j) \notin \mathcal{L}_0$. Thus, $\mathcal{B}_0$ wins its soundness experiment.

So one of the two algorithms $\mathcal{B}_0$ or $\mathcal{B}_1$ wins its soundness experiment with the same probability $\epsilon(\lambda)$ as $\mathcal{A}$ wins its own, which is non-negligible.

### B.4 Proof of Theorem 4

Throughout this proof, we will refer to $(\mathbb{G}, g, p)$ as the group $\mathbb{G}$ of prime order $p$ generated by $g$ used in $\mathsf{ElGamal}$. $\mathsf{CC2}$ uses two $\mathsf{NIP}$ in the algorithm $\mathsf{Play}$ (one for the case $A = 1$, the other for the case $A = 0$). Since these proofs are assumed to be zero-knowledge, there exists two simulators $\mathrm{Sim}_0$ and $\mathrm{Sim}_1$ that simulates valid proofs from any statment. We build the following cryptid simulator.

$\mathrm{Sim}(\mathrm{set}, t, \mathrm{map}, \alpha^*, \mathrm{clue}_{\alpha^*}, \{(\alpha_i, A_i, j_i)\}_{i\in[\![t]\!]^*})$**:** runs $(\mathsf{pc}_{\alpha^*}, \mathsf{sc}_{\alpha^*}) \leftarrow \mathsf{GenClue}(\mathrm{set}, \mathrm{clue}_{\alpha^*})$ and parses $\mathsf{pc}_{\alpha^*}$ as $(\mathsf{pk}_{\alpha^*}, E_{\alpha^*,1}, E_{\alpha^*,2}, E_{\alpha^*,3})$. For each $\alpha \in [\![n]\!]^* \backslash \{\alpha^*\}$, picks $(\mathsf{pk}_\alpha, \gamma_{\alpha,1}, \gamma_{\alpha,2}, \gamma_{\alpha,3}) \xleftarrow{\$} \mathbb{G}^4$, for all $i \in [\![3]\!]^*$ computes $E_{\alpha,i} \leftarrow \mathsf{Enc}_{\mathsf{pk}_\alpha}(\gamma_{\alpha,i})$, then computes $\mathsf{pc}_\alpha \leftarrow (\mathsf{pk}_\alpha, E_{\alpha,1}, E_{\alpha,2}, E_{\alpha,3})$. It parses map as $(\mathsf{cell}_i)_{i\in[\![N]\!]^*}$ and $\mathsf{cell}_j$ as $(T_j, \mathcal{P}_j)$. According to the rules of Cryptid, the simulator simulates each play $i$ for player $\mathrm{Player}_{\alpha_i}$ (from play 1 to play $t$) by computing $\pi_i \leftarrow \mathrm{Sim}_{A_i}(\mathsf{ElGamal}, \mathsf{pk}_{\alpha_i}, E_{\alpha_i,1}, E_{\alpha_i,2}, E_{\alpha_i,3}, T_{j_i}, \mathcal{P}_{j_i})$. Finally, it returns $\mathsf{view}_{\alpha^*} = (\mathrm{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i\in[\![t]\!]^*})$.

Note that this simulator replaces each ciphertext with the encryption of a random element and simulates each $\mathsf{NIP}$ .

To prove this theorem, we use the following sequence of games [21] where the first game matches the case where the distinguisher receives elements generated by the real protocol (case $b = 0$ in the confidentiality experiment), and where the last game matches the case where the distinguisher receives elements generated by the simulator (case $b = 1$ in the confidentiality experiment). By showing that each of the intermediate games are computationally indistinguishable from the other, we deduce that the two cases are indistinguishable.

**Game** $G_0$**:** $\mathcal{D}$ receives as input $(\mathrm{set}, t, \mathrm{map}, \alpha^*, \mathrm{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i\in[\![t]\!]^*})$ as in the case $b = 0$ (corresponding to the real protocol) in the confidentiality experiment. we have:

$$\Pr[\mathcal{D} \text{ returns } 1 \text{ in } G_0] = \Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{CC2},\mathcal{D},0}(\lambda)\right].$$

**Game** $G_1$**:** This game is the same as $G_0$ except that each proof $\pi_i$ is replaced by a simulated proof. Since all $\mathsf{NIP}$ are zero-knowledge, we have:

$$\Pr[\mathcal{D} \text{ returns } 1 \text{ in } G_0] = \Pr[\mathcal{D} \text{ returns } 1 \text{ in } G_1].$$

At this step, the only difference between $G_1$ and the case $b = 1$ in the confidentiality experiment (corresponding to the simulator) is that in the simulator, ciphertexts are generated from random messages.

**Game** $G_2$**:** $\mathcal{D}$ receives as input $(\mathrm{set}, t, \mathrm{map}, \alpha^*, \mathrm{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i\in[\![t]\!]^*})$ as in the case $b = 1$ (corresponding to the simulator) in the confidentiality experiment. we have:

$$\Pr[\mathcal{D} \text{ returns } 1 \text{ in } G_2] = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{CC2},\mathcal{D},1}(\lambda)\right].$$

To prove indistinguishability between $G_1$ and $G_2$, we use an hybrid argument [14]. We consider a sequence of games from $G_1$ to $G_2$ in which we progressively replace each ciphertext in $G_1$ with the encryption of a random element.

**Game $G_{1,\alpha}$ (for all $\alpha \in [\![n]\!]^*$):** We define $G_{1,0}$ as $G_1$, and $G_{1,n}$ as $G_2$. The game $G_{1,\alpha}$ is the same as $G_{1,\alpha-1}$ except that if $\alpha \neq \alpha^*$, then the challenger picks $(\mathsf{pk}_\alpha, \gamma_{\alpha,1}, \gamma_{\alpha,2}, \gamma_{\alpha,3}) \xleftarrow{\$} \mathbb{G}^4$, runs $E_{\alpha,l} = \mathsf{Enc}_{\mathsf{pk}_\alpha}(\gamma_{\alpha,l})$ for each $l \in [\![3]\!]^*$, and sets $\mathsf{pc}_\alpha = (\mathsf{pk}_\alpha, E_{\alpha,1}, E_{\alpha,2}, E_{\alpha,3})$.

**Game $G_{1,\alpha,l}$ (for all $l \in [\![3]\!]^*$):** We define $G_{1,\alpha,0}$ as $G_{1,\alpha-1}$ and $G_{1,\alpha,3}$ as $G_{1,\alpha}$. The game $G_{1,\alpha,l}$ is the same as $G_{1,\alpha,l-1}$ except that if $\alpha \neq \alpha^*$, then it computes $\gamma_{\alpha,l} \xleftarrow{\$} \mathbb{G}$ and sets $E_{\alpha,l} \leftarrow \mathsf{Enc}_{\mathsf{pk}_\alpha}(\gamma_{\alpha,l})$.

We emphasize that $G_{1,\alpha^*,l-1} = G_{1,\alpha^*,l}$. Let $\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda)$ be the IND-CPA advantage of the encryption scheme ElGamal. We claim that, for any $\alpha \in [\![n]\!]^* \backslash \{\alpha^*\}$ and $l \in [\![3]\!]^*$:

$$\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) \geq \left|\Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l-1}] - \Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l}]\right|.$$

We prove this claim by reduction. We build the following two-party p.p.t. algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the IND-CPA experiment on ElGamal:

$\mathcal{A}_1(\mathsf{pk})$**:** sets $\mathsf{pk}_\alpha \leftarrow \mathsf{pk}$, parses $\mathrm{clue}_\alpha$ as $(C_{\alpha,1}, C_{\alpha,2}, C_{\alpha,3})$, sets $m_0 = C_{\alpha,l}$ and picks $m_1 \xleftarrow{\$} \mathbb{G}$, and returns $(m_0, m_1)$.

$\mathcal{A}_2(c)$**:** sets $E_{\alpha,l} = c$, generates a tuple $\mathsf{input} = (\mathrm{set}, t, \mathrm{map}, \alpha^*, \mathrm{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha \in [\![n]\!]^*}, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i \in [\![t]\!]*})$ as in $G_{1,\alpha,l-1}$ except that it uses its own values $\mathsf{pk}_\alpha$ and $E_{\alpha,l}$ in $\mathsf{pc}_\alpha$, and runs $b' \leftarrow \mathcal{D}(\mathsf{input})$. It returns $b'$.

We have:

$$\Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},0}(\lambda)\right] = \Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l-1}],$$

$$\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},1}(\lambda)\right] = \Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l}].$$

Thus, we deduce:

$$\begin{aligned}\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) &\geq \left|\Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},0}(\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},1}(\lambda)\right]\right| \\ &= \left|\Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l-1}] - \Pr[\mathcal{D} \text{ returns 1 in } G_{1,\alpha,l}]\right|,\end{aligned}$$

which concludes the proof of the claim. We deduce that:

$$3(n-1)\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) \geq \left|\Pr[\mathcal{D} \text{ returns 1 in } G_1] - \Pr[\mathcal{D} \text{ returns 1 in } G_2]\right|.$$

Finally:

$$\begin{aligned}3(n-1)\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) &\geq \left|\Pr[\mathcal{D} \text{ returns 1 in } G_0] - \Pr[\mathcal{D} \text{ returns 1 in } G_2]\right| \\ &= \left|\Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{CC2},\mathcal{A},0}(\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{CC2},\mathcal{A},1}(\lambda)\right]\right|,\end{aligned}$$

which concludes the proof of the theorem, since $3(n-1)\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda)$ is negligible.

## C Benchmarks of CC1 and CC2

To demonstate the practicality of our schemes, we implement the algorithms GenClue, OpenClue, Play, and Verify in Rust. We use the prime order group Ristretto with the curve25519_dalek [18] library and we use 256 bits secret keys. The source code for our implementation is available at `https://anonymous.4open.science/r/cryptidscheme`. Table 1 describes the average computation time, measured in milliseconds, of each algorithm for our two cryptographic crytid schemes, and Table 2 illustrates the average size of the public clue key pc and the size of the proof $\pi$ generated by the players. The averages are based on 500 runs. Our measurements depend on the player's answer $\{\texttt{maybe}, \texttt{no}\}$, moreover for the scheme CC2, we differentiate between the two forms of clue. Our experiments are not based on the real maps proposed in the original game, but on randomly generated cells and clues. We have chosen to evaluate our algorithms on cells that match all the properties together (or all but one in the case where the clue concerns a neighbouring property and the answer is `no`), as this maximises the generation time and the size of the proof in CC2 (it makes no difference for CC1). The times given are therefore worst-case.

As we had predicted in the theoretical analysis of our schemes, the computation time and the size of the public clues are higher in CC1 than in CC2, conversely the computation time and the size of the proofs are higher in CC2. The generation of all public clues is done once by the game editor, and each occurrence of OpenClue, Play, and Verify is done by the players. Generating/verifying a clue or building a proof takes less than 300 milliseconds in all cases, which is reasonable for playing the game in real time. Opening a clue takes longer in CC1 (less than 500 milliseconds versus 13 in CC2), but is still efficient and only needs to be done once per player at the start of the game. On the other hand, a Cryptid game consists of 180 game setup for 3 to 5 players, so its cryptographic version requires the download of $180 \times 4 = 720$ public clue keys, *i.e.*, about $400 \times 720 = 288000$ kilobytes for CC1 and $13 \times 720 = 9360$ kilobytes for CC2. In summary, CC1 requires more memory to download and store when acquiring the game, but has a lower latency during gameplay.

## D Security Proofs of VCC

In this section, we give the security proofs of our extended scheme.

### D.1 Proof of Theorem 7

We define the algorithm $\mathsf{OpenClue}^*$ for VCC as follows:

$\mathsf{OpenClue}^*(\mathsf{pc})$**:** parses pc as $(\mathsf{pk}, (E_i)_{i\in[\![3]\!]^*})$, computes $\mathsf{sc} \leftarrow \log_g(\mathsf{pk})$, and returns the output of $\mathsf{OpenClue}(\mathsf{pc}, \mathsf{sc})$.

Note that $\forall(\mathsf{pc}, \mathsf{sc}) \leftarrow \mathsf{GenClue}(\text{clue})$, we have $\mathsf{OpenClue}(\mathsf{pc}, \mathsf{sc}) = \mathsf{OpenClue}^*(\mathsf{pc})$ because $\mathsf{pk} = g^{\mathsf{sc}}$.

**Table 1.** Running time in milliseconds for our two cryptographic cryptid scheme. The results are an average over 500 executions.

| Scheme | CC1 | | CC2 | | | |
|---|---|---|---|---|---|---|
| Answer | `no` | `maybe` | `no` | | `maybe` | |
| clue | $(\text{clue}_i)_{i\in[\![N]\!]^*} \in \{0,1\}^N$ | | $(\bar{T}_1,\bar{T}_2,\perp)$ | $(\perp,\perp,\bar{P})$ | $(\bar{T}_1,\bar{T}_2,\perp)$ | $(\perp,\perp,\bar{P})$ |
| GenClue | 867.80 | 867.80 | 28.47 | 28.34 | 27.79 | 28.56 |
| OpenClue | 434.71 | 434.71 | 12.47 | 12.48 | 12.19 | 12.47 |
| Play | 8.17 | 8.09 | 280.49 | 264.91 | 140.69 | 173.78 |
| Verify | 16.11 | 16.08 | 213.93 | 202.00 | 143.60 | 143.05 |

**Table 2.** Size in kilobytes for our two cryptographic cryptid scheme. The results are an average over 500 executions.

| Scheme | CC1 | | CC2 | | | |
|---|---|---|---|---|---|---|
| Answer | `no` | `maybe` | `no` | | `maybe` | |
| clue | $(\text{clue}_i)_{i\in[\![N]\!]^*} \in \{0,1\}^N$ | | $(\bar{T}_1,\bar{T}_2,\perp)$ | $(\perp,\perp,\bar{P})$ | $(\bar{T}_1,\bar{T}_2,\perp)$ | $(\perp,\perp,\bar{P})$ |
| pc | 398.51 | 398.51 | 12.86 | 12.85 | 12.85 | 12.86 |
| $\pi$ | 3.86 | 3.86 | 94.08 | 88.56 | 36.75 | 36.74 |

Let $\lambda$ be a security parameter. We will show that the probability that any p.p.t. algorithm $\mathcal{A}$ wins the game-soundness experiment on VCC is negligible. We use the following sequence of games [21] where the first game is the game-soundness experiment.

**Game $G_0$:** This game corresponds to the game-soundness experiment, we have:

$$\Pr[\mathcal{A} \text{ wins } G_0] = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Game\text{-}Soundness}}_{\mathsf{VCC},\mathcal{A}}(\lambda)\right].$$

**Game $G_1$:** This game is the same as $G_0$ except that aborts and returns 0 on the event $F_1$ = "one element is associated with $1_{\mathbb{G}}$ during the setup process". During the setup, each element of $\mathcal{T} \cup \mathcal{P} \cup \{\perp\}$ is associated with a group element picking in the uniform distribution on $\mathbb{G}$. Since in VCC we have $|\mathcal{T} \cup \mathcal{P} \cup \{\perp\}| = 20$, we deduce that:

$$20/p \geq \Pr[F_1] \geq |\Pr[\mathcal{A} \text{ wins in } G_0] - \Pr[\mathcal{A} \text{ wins } G_1]|.$$

**Game $G_2$:** This game is the same as $G_1$ except that aborts and returns 0 on the event $F_2$ = "Two different elements are associated with the same group element during the setup process". Since 20 group elements are picked in the uniform distribution on $\mathbb{G}$, we have:

$$20^2/p = 400/p \geq \Pr[F_2] \geq |\Pr[\mathcal{A} \text{ wins in } G_1] - \Pr[\mathcal{A} \text{ wins } G_2]|.$$

At this step, each group element associated with one element of $\mathcal{T} \cup \mathcal{P} \cup \{\perp\}$ has been picked in the uniform distribution on $\mathbb{G}$.

**Game $G_3$:** This game is the same as $G_2$ except that $G_3$ uses the extractors on each of the $2n+2$ proofs $\rho_0, \rho_1$, $(\rho_{2,\alpha})_{\alpha\in[\![n]\!]^*}$, and $(\rho_{3,\alpha})_{\alpha\in[\![n]\!]^*}$, and aborts and returns 0 on the event $F_3$ = "An extractor fails on at least one proof".

Let $\epsilon_{\mathsf{ext}}(\lambda)$ be the $\mathsf{max}$ on the failure probability of the extractors, we have:

$$(2n+2)\epsilon_{\mathsf{ext}}(\lambda) \geq \Pr[F_3] \geq \left|\Pr[\mathcal{A} \text{ wins } G_2] - \Pr[\mathcal{A} \text{ wins } G_3]\right|.$$

$G_3$ extracts the witnesses $\mathsf{sk}$ from the proofs $\rho_0, \rho_1$, and $(\rho_{2,\alpha})_{\alpha\in[\![n]\!]^*}$, and the witnesses $(\mathsf{sk}_\alpha)_{\alpha\in[\![n]\!]^*}$ from the proofs $(\rho_{2,\alpha})_{\alpha\in[\![n]\!]^*}$ and $(\rho_{3,\alpha})_{\alpha\in[\![n]\!]^*}$. We note that if $\mathcal{A}$ wins its game, then these witnesses are correctly extracted, and all the extracted witnesses $\mathsf{sk}$ (resp. $\mathsf{sk}_\alpha$ for all $\alpha \in [\![n]\!]^*$) are the same, because each proof implicitly uses the same $\mathsf{pk} = g^{\mathsf{sk}}$ (resp. $\mathsf{pk}_\alpha = g^{\mathsf{sk}_\alpha}$, where $\mathsf{pk}_\alpha$ is the first element of $\mathsf{pc}_\alpha$). According to the definition of $\mathsf{OpenClue}^*$, for each $\alpha \in [\![n]\!]^*$ we have $\mathrm{clue}_\alpha = \mathsf{OpenClue}(\mathsf{pc}_\alpha, \mathsf{sk}_\alpha) = \mathsf{OpenClue}^*(\mathsf{pc}_\alpha)$. For each $\alpha \in [\![n]\!]^*$, we parse $\mathrm{clue}_\alpha$ as $(C_{\alpha,i})_{i\in[\![3]\!]^*}$. If the proofs $(\rho_{3,\alpha})_{\alpha\in[\![n]\!]^*}$ are correctly extracted, then for each $\alpha \in [\![n]\!]^*$:

$$\begin{pmatrix} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,3}) = \bot \wedge \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,1}) \neq \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,2}) \\ \wedge \left(\bigvee_{\bar{T}\in\mathcal{T}} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,1}) = \bar{T}\right) \wedge \left(\bigvee_{\bar{T}\in\mathcal{T}} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,2}) = \bar{T}\right) \end{pmatrix}$$
$$\vee \left(\mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,1}) = \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,2}) = \bot \wedge \left(\bigvee_{\bar{P}\in\mathcal{P}} \mathsf{Dec}_{\mathsf{sk}_\alpha}(E_{\alpha,3}) = \bar{P}\right)\right).$$

We deduce that $((C_{\alpha,1}, C_{\alpha,2}) \in \mathcal{T}^2$ and $C_{\alpha,1} \neq C_{\alpha,2}$ and $C_{\alpha,3} = \bot))$ or $(C_{\alpha,1} = C_{\alpha,1} = \bot$ and $C_{\alpha,3} \in \mathcal{P})$. For all $\alpha \in [\![n]\!]^*$, we have $1 = \mathsf{CorrectClue}(\mathrm{map}, \mathrm{clue}_\alpha)$.

**Game $G_4$:** We parse map as $(\mathrm{cell}_i)_{i\in[\![N]\!]^*}$, each $\mathrm{cell}_i$ as $(T_i, \mathcal{P}_i)$, and the second element of $\rho$ as $(E_l)_{l\in[\![14]\!]}$. This game is the same as $G_3$ except that $G_4$ aborts and returns 0 on the event $F_4 =$ " $\exists i \in [\![N]\!]^*;\ \mathsf{Dec}_{\mathsf{sk}}\left(\prod_{l\in[\![14]\!]^*} E_l\right) = \prod_{\bar{P}_l\in\mathcal{P}_i} \bar{P}_l$ $\wedge\ \exists l \in [\![14]\!]^*;\ ((\mathsf{Dec}_{\mathsf{sk}}(E_l) = 1 \wedge \bar{P}_l \in \mathcal{P}_i) \vee (\mathsf{Dec}_{\mathsf{sk}}(E_l) = \bar{P}_l \wedge \bar{P}_l \notin \mathcal{P}_i))$", where $\mathsf{sk}$ is the witness extracted from the game $G_3$. We have:

$$\Pr[F_4] \geq \left|\Pr[\mathcal{A} \text{ wins } G_3] - \Pr[\mathcal{A} \text{ wins } G_4]\right|.$$

We claim that:

$$\epsilon_{14-\mathsf{col}}(\lambda) \geq \Pr[F_4],$$

where $\epsilon_{14-\mathsf{col}}(\lambda)$ is defined in Theorem 5. We prove this claim by reduction. We build the following p.p.t. algorithm $\mathcal{B}$ playing the experiment defined in Theorem 5:

$\mathcal{B}\left((g_l)_{l\in[\![14]\!]^*}\right)$**:** simulates the game $G_3$ to $\mathcal{A}$, except that during the setup generation, $\mathcal{B}$ associates each $g_l$ at each $\bar{P}_l \in \mathcal{P}$, and aborts if one $g_l = 1_{\mathbb{G}}$ for some $l$. This does not disturb the simulation of $G_3$ since, in accordance with the rules of $G_2$, the elements associated with the $\mathcal{P}$ elements come from the uniform ditribution on $\mathbb{G}$. It parses each $\mathrm{cell}_i$ as $(T_i, \mathcal{P}_i)$. It runs $(\rho, \mathrm{map}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}) \leftarrow \mathcal{A}(\mathrm{set})$. If $F_4$ does not occur, then $\mathcal{B}$ aborts its experiment, else it parses the second element of $\rho$ as $(E_l)_{l\in[\![14]\!]}$ and extracts $\mathsf{sk}$ from $\rho$. For each $l \in [\![14]\!]^*$, it computes $V_l \leftarrow \mathsf{Dec}_{\mathsf{sk}}(E_l)$.

$\mathcal{B}$ finds $i \in [\![N]\!]^*$ s.t. $\prod_{l \in [\![14]\!]^*} V_l = \prod_{\bar{P}_l \in \mathcal{P}_i} \bar{P}_l$ and there exists $l \in [\![14]\!]^*$ s.t. ($V_l = 1$ and $\bar{P}_l \in \mathcal{P}_i$) or ($V_l = \bar{P}_l$ and $\bar{P}_l \notin \mathcal{P}_i$). Note that such a $i$ exists since $F_4$ occurred. For each $l \in [\![14]\!]^*$, if $V_l = \bar{P}_l$, then $\mathcal{B}$ sets $x_l \leftarrow 1$, else it sets $x_l \leftarrow 0$, and if $\bar{P}_l \in \mathcal{P}_i$, then $\mathcal{B}$ sets $y_l \leftarrow 1$, else it set $y_l \leftarrow 0$.
It sets $x = (x_l)_{l \in [\![14]\!]^*}$ and $y = (y_l)_{l \in [\![14]\!]^*}$, then returns $(x, y)$. Note that since $F_4$ occurred, then $x \neq y$, and $\mathcal{B}$ returns $(x, y)$.

If $\mathcal{A}$ wins $G_3$ and $F_4$ occurs, then $\mathcal{B}$ returns $x, y \in \{0,1\}^l$ s.t. $x \neq y$ and $\prod_{l=1}^{14} g_l^{x_l} = \prod_{l=1}^{14} g_l^{y_l}$, which happens with negligible probability $\epsilon_{14-\mathsf{col}}(\lambda)$ under the discrete logarithm assumption according to Theorem 5. We deduce that:

$$\epsilon_{14-\mathsf{col}}(\lambda) \geq \Pr[F_4] \geq |\Pr[\mathcal{A} \text{ wins } G_3] - \Pr[\mathcal{A} \text{ wins } G_4]|\,,$$

which concludes the proof of the claim.

At this step, we show that $\mathcal{A}$ has no way to winning because if $G_4$ does not abort (*i.e.*, $F_1$, $F_2$, $F_3$, and $F_4$ does not occur), then there exists $i \in [\![N]\!]^*$ s.t. for all $\alpha \in [\![n]\!]^*$, we have $1 = \mathsf{Answer}(\mathrm{map}, \mathrm{clue}_\alpha, i)$:
Assume that $G_4$ does not abort. For all $l \in [\![14]\!]$, we set $V_l = \mathsf{Dec}_{\mathsf{sk}}(E_l)$. $\rho_0$ and $\rho_1$ are valid proofs (since $F_3$ does not occur), and there exists $i \in [\![N]\!]^*$ s.t. $V_0 = T_i$ and $\prod_{l \in [\![14]\!]^*} V_l = \prod_{\bar{P}_l \in \mathcal{P}_i} \bar{P}_l$ and $\forall l \in [\![14]\!]^*((V_l = 1$ and $\bar{P}_l \notin \mathcal{P}_i)$ or $(V_l = \bar{P}_l \in \mathcal{P}_i))$ (since $F_4$ does not occur).
For each $\alpha \in [\![n]\!]^*$, $\rho_{2,\alpha}$ is a valid proof (since $F_3$ does not occur), so $C_{\alpha,1} = V_0$ or $C_{\alpha,2} = V_0$ or there exists $l \in [\![14]\!]^*$ s.t. $C_{\alpha,3} = V_l$.
We deduce that there exists $i \in [\![N]\!]^*$ s.t. for each $\alpha \in [\![n]\!]^*$, we have $C_{\alpha,1} = T_i$ or $C_{\alpha,2} = T_i$ or there exists $l \in [\![14]\!]^*$ s.t. if $\bar{P}_l \in \mathcal{P}_i$, then $C_{\alpha,3} = V_l = \bar{P}_l$, else $C_{\alpha,3} = V_l = 1_{\mathbb{G}}$.
For each $\alpha \in [\![n]\!]^*$, $\rho_{\alpha,3}$ is a valid proof, so $1 = \mathsf{CorrectClue}(\mathrm{map}, \mathrm{clue}_\alpha)$. We have for each $\alpha \in [\![n]\!]^*$, ($C_{\alpha,1} = \perp$ and $C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$) or ($C_{\alpha,3} = \perp$ and there exists $(l_1, l_2) \in [\![5]\!]^*$ s.t. $\bar{T}_{l_1} \neq \bar{T}_{l_2}$, and s.t. $C_{\alpha,1} = \bar{T}_{l_1}$, and $C_{\alpha,2} = \bar{T}_{l_2}$).
This implies that there exists $i \in [\![N]\!]^*$ s.t. for each $\alpha \in [\![n]\!]^*$, ($C_{\alpha,1} = T_i$ or $C_{\alpha,2} = T_i$ or there exists $l \in [\![14]\!]^*$ s.t. if $\bar{P}_l \in \mathcal{P}_i$, then $C_{\alpha,3} = V_l = \bar{P}_l$, else $C_{\alpha,3} = V_l = 1_{\mathbb{G}}$), and (($C_{\alpha,1} = \perp$ and $C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$) or ($C_{\alpha,3} = \perp$ and there exists $(l_1, l_2) \in [\![5]\!]^*$ s.t. $\bar{T}_{l_1} \neq \bar{T}_{l_2}$, and s.t. $C_{\alpha,1} = \bar{T}_{l_1}$, and $C_{\alpha,2} = \bar{T}_{l_2}$)).
We deduce that there exists $i \in [\![N]\!]^*$ s.t. for each $\alpha \in [\![n]\!]^*$:

- Either $C_{\alpha,1} = T_i$ and $C_{\alpha,2} = T_i$ and there exists $l \in [\![14]\!]^*$ s.t. $C_{\alpha,3} = V_l = \bar{P}_l$ if $\bar{P}_l \in \mathcal{P}_i$, $1_{\mathbb{G}}$ else, and $C_{\alpha,1} = \perp$ and $C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. We have $T_i = \perp$, which is contradicting because the event $F_2$ does not occur, so this case does not occur.
- Either $C_{\alpha,1} = T_i$ and $C_{\alpha,2} = T_i$ and there exists $l \in [\![14]\!]^*$ s.t. if $\bar{P}_l \in \mathcal{P}_i$, then $C_{\alpha,3} = V_l = \bar{P}_l$, else $C_{\alpha,3} = 1_{\mathbb{G}}$, and $C_{\alpha,3} = \perp$ and there exists $(l_1, l_2) \in [\![5]\!]^*$, s.t. $\bar{T}_{l_1} \neq \bar{T}_{l_2}$ and $C_{\alpha,1} = \bar{T}_{l_1}$ and $C_{\alpha,2} = \bar{T}_{l_2}$. We have $\bar{T}_{l_1} = T_i$ and $\bar{T}_{l_2} = T_i$ and $\bar{T}_{l_1} \neq \bar{T}_{l_2}$, which is contradicting, so this case does not occur.

- Either $C_{\alpha,1} = T_i$ and $C_{\alpha,2} = T_i$ and for all $l \in [\![14]\!]^*$, $C_{\alpha,3} \neq V_l$ and if $\bar{P}_l \in \mathcal{P}_i$, then $V_l = \bar{P}_l$, else $V_l = 1_{\mathbb{G}}$, and $C_{\alpha,1} = \perp$ and $C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. We have $T_i = \perp$, which is contradicting because $F_2$ does not occur, so this case does not occur.
- Either $C_{\alpha,1} = T_i$ and $C_{\alpha,2} = T_i$ and for all $l \in [\![14]\!]^*$, $C_{\alpha,3} \neq V_l$ and if $\bar{P}_l \in \mathcal{P}_i$, then $V_l = \bar{P}_l$, else $V_l = 1_{\mathbb{G}}$, and $C_{\alpha,3} = \perp$ and there exists $(l_1, l_2) \in [\![5]\!]^*$, s.t. $\bar{T}_{l_1} \neq \bar{T}_{l_2}$ and $C_{\alpha,1} = \bar{T}_{l_1}$ and $C_{\alpha,2} = \bar{T}_{l_2}$. We have $\bar{T}_{l_1} = T_i$ and $\bar{T}_{l_2} = T_i$ and $\bar{T}_{l_1} \neq \bar{T}_{l_2}$, which is contradicting, so this case does not occur.
- Either $C_{\alpha,1} = T_i$ and $C_{\alpha,2} \neq T_i$ and for all $l \in [\![14]\!]^*$, $C_{\alpha,3} \neq V_l$ and if $\bar{P}_l \in \mathcal{P}_i$, then $V_l = \bar{P}_l$, else $V_l = 1_{\mathbb{G}}$, and $C_{\alpha,1} = \perp$ and $C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. We have $T_i = \perp$, which is contradicting because $F_2$ does not occur, so this case does not occur.
- Either $C_{\alpha,1} \neq T_i$ and $C_{\alpha,2} = T_i$ and for all $l \in [\![14]\!]^*$, $C_{\alpha,3} \neq V_l$ and if $\bar{P}_l \in \mathcal{P}_i$, then $V_l = \bar{P}_l$, else $V_l = 1_{\mathbb{G}}$, and $C_{\alpha,1} = \perp$ and $C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. We have $T_i = \perp$, which is contradicting because $F_2$ does not occur, so this case does not occur.
- Either $C_{\alpha,1} = T_i$ and $C_{\alpha,2} \neq T_i$ and for all $l \in [\![14]\!]^*$, $C_{\alpha,3} \neq V_l$ and if $\bar{P}_l \in \mathcal{P}_i$, then $V_l = \bar{P}_l$, else $V_l = 1_{\mathbb{G}}$, and $C_{\alpha,3} = \perp$ and there exists $(l_1, l_2) \in [\![5]\!]^*$, s.t. $\bar{T}_{l_1} \neq \bar{T}_{l_2}$ and $C_{\alpha,1} = \bar{T}_{l_1}$ and $C_{\alpha,2} = \bar{T}_{l_2}$. $F_1$ and $F_2$ do not occur, we have $\bar{T}_{l_1} \neq \bar{T}_{l_2}$ and for all $l \in [\![14]\!]^*$, we have $\bar{P}_l \neq \perp$, and $\perp \neq 1_{\mathbb{G}}$. Then, we have $C_{\alpha,1} = \bar{T}_{l_1} = T_i$ and $C_{\alpha,2} = \bar{T}_{l_2} \neq T_i$ and $(\bar{T}_{l_1}, \bar{T}_{l_2}) \in \mathcal{T}^2$ and $C_{\alpha,3} = \perp$.
- Either $C_{\alpha,1} \neq T_i$ and $C_{\alpha,2} = T_i$ and for all $l \in [\![14]\!]^*$, $C_{\alpha,3} \neq V_l$ and if $\bar{P}_l \in \mathcal{P}_i$, then $V_l = \bar{P}_l$, else $V_l = 1_{\mathbb{G}}$, and $C_{\alpha,3} = \perp$ and there exists $(l_1, l_2) \in [\![5]\!]^*$, s.t. $\bar{T}_{l_1} \neq \bar{T}_{l_2}$ and $C_{\alpha,1} = \bar{T}_{l_1}$ and $C_{\alpha,2} = \bar{T}_{l_2}$. Since $F_1$ and $F_2$ do not occur, we have $\bar{T}_{l_1} \neq \bar{T}_{l_2}$ and for all $l \in [\![14]\!]^*$, we have $\bar{P}_l \neq \perp$, and $\perp \neq 1_{\mathbb{G}}$. Then, we have $C_{\alpha,1} = \bar{T}_{l_1} \neq T_i$ and $C_{\alpha,2} = \bar{T}_{l_2} = T_i$ and $(\bar{T}_{l_1}, \bar{T}_{l_2}) \in \mathcal{T}^2$ and $C_{\alpha,3} = \perp$.
- Either $C_{\alpha,1} \neq T_i$ and $C_{\alpha,2} \neq T_i$ and there exists $l \in [\![14]\!]^*$ s.t. if $\bar{P}_l \in \mathcal{P}_i$, then $C_{\alpha,3} = V_l = \bar{P}_l$, else $C_{\alpha,3} = 1_{\mathbb{G}}$, and $C_{\alpha,1} = \perp$ and $C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. This implies that $C_{\alpha,1} = C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. Since $F_1$ does not occur, $\bar{P} \neq 1_{\mathbb{G}}$, then $\bar{P} = \bar{P}_l \in \mathcal{P}_i$.
- Either $C_{\alpha,1} \neq T_i$ and $C_{\alpha,2} \neq T_i$ and there exists $l \in [\![14]\!]^*$ s.t. if $\bar{P}_l \in \mathcal{P}_i$, then $C_{\alpha,3} = V_l = \bar{P}_l$, else $C_{\alpha,3} = 1_{\mathbb{G}}$, and $C_{\alpha,3} = \perp$ and there exists $(l_1, l_2) \in [\![5]\!]^*$, s.t. $\bar{T}_{l_1} \neq \bar{T}_{l_2}$ and $C_{\alpha,1} = \bar{T}_{l_1}$ and $C_{\alpha,2} = \bar{T}_{l_2}$. We have $\bar{P}_l = \perp$ or $1_{\mathbb{G}} = \perp$, which is contradicting because $F_1$ and $F_2$ do not occur, so this case does not occur.
- Either $C_{\alpha,1} = T_i$ and $C_{\alpha,2} \neq T_i$ and there exists $l \in [\![14]\!]^*$ s.t. if $\bar{P}_l \in \mathcal{P}_i$, then $C_{\alpha,3} = V_l = \bar{P}_l$, else $C_{\alpha,3} = 1_{\mathbb{G}}$, and $C_{\alpha,1} = \perp$ and $C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. This implies that $C_{\alpha,1} = C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. We have $T_i = \perp$, which is contradicting because the event $F_2$ does not occur, so this case does not occur.
- Either $C_{\alpha,1} \neq T_i$ and $C_{\alpha,2} = T_i$ and there exists $l \in [\![14]\!]^*$ s.t. if $\bar{P}_l \in \mathcal{P}_i$, then $C_{\alpha,3} = V_l = \bar{P}_l$, else $C_{\alpha,3} = 1_{\mathbb{G}}$, and $C_{\alpha,1} = \perp$ and $C_{\alpha,2} = \perp$ and

there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. This implies that $C_{\alpha,1} = C_{\alpha,2} = \perp$ and there exists $\bar{P} \in \mathcal{P}$ s.t. $C_{\alpha,3} = \bar{P}$. We have $T_i = \perp$, which is contradicting because the event $F_2$ does not occur, so this case does not occur.
- Either $C_{\alpha,1} = T_i$ and $C_{\alpha,2} \neq T_i$ and there exists $l \in [\![14]\!]^*$ s.t. if $\bar{P}_l \in \mathcal{P}_i$, then $C_{\alpha,3} = V_l = \bar{P}_l$, else $C_{\alpha,3} = 1_{\mathbb{G}}$, and $C_{\alpha,3} = \perp$ and there exists $(l_1, l_2) \in [\![5]\!]^*$, s.t. $\bar{T}_{l_1} \neq \bar{T}_{l_2}$ and $C_{\alpha,1} = \bar{T}_{l_1}$ and $C_{\alpha,2} = \bar{T}_{l_2}$. We have $\bar{P}_l = \perp$ or $1_{\mathbb{G}} = \perp$, which is contradicting because $F_1$ and $F_2$ do not occur, so this case does not occur.
- Otherwise $C_{\alpha,1} \neq T_i$ and $C_{\alpha,2} = T_i$ and there exists $l \in [\![14]\!]^*$ s.t. if $\bar{P}_l \in \mathcal{P}_i$, then $C_{\alpha,3} = V_l = \bar{P}_l$, else $C_{\alpha,3} = 1_{\mathbb{G}}$, and $C_{\alpha,3} = \perp$ and there exists $(l_1, l_2) \in [\![5]\!]^*$, s.t. $\bar{T}_{l_1} \neq \bar{T}_{l_2}$ and $C_{\alpha,1} = \bar{T}_{l_1}$ and $C_{\alpha,2} = \bar{T}_{l_2}$. We have $\bar{P}_l = \perp$ or $1_{\mathbb{G}} = \perp$, which is contradicting because $F_1$ and $F_2$ do not occur, so this case does not occur.

Finally there exists $i \in [\![N]\!]^*$ s.t. for each $\alpha \in [\![n]\!]^*$, $(C_{\alpha,1}, C_{\alpha,2}, C_{\alpha,3}) = (T_i, \bar{T}_{l_2}, \perp)$ where $(T_i, \bar{T}_{l_2}) \in \mathcal{T}^2$ and $T_i \neq \bar{T}_{l_2}$ or $(C_{\alpha,1}, C_{\alpha,2}, C_{\alpha,3}) = (\bar{T}_{l_1}, T_i, \perp)$ where $(T_i, \bar{T}_{l_1}) \in \mathcal{T}^2$ and $T_i \neq \bar{T}_{l_1}$ or $(C_{\alpha,1}, C_{\alpha,2}, C_{\alpha,3}) = (\perp, \perp, \bar{P}_l)$ where $\bar{P}_l \in \mathcal{P}_i$.
This implies that there exists $i \in [\![N]\!]^*$ s.t. for each $\alpha \in [\![n]\!]^*$, $1 = \mathsf{Answer}(\text{map}, \text{clue}_\alpha, i)$, which implies that $\mathcal{A}$ cannot win $G_4$ even if $G_4$ does not abort.

Finally, we have:

$$\frac{420}{p} + (2n+2)\epsilon_{\mathsf{ext}}(\lambda) + \epsilon_{14-\mathsf{col}}(\lambda) \geq \left|\Pr[\mathcal{A} \text{ wins } G_0] - \Pr[\mathcal{A} \text{ wins } G_4]\right|,$$

which concludes the proof of the theorem, since $\frac{420}{p}$, $(2n+2)\epsilon_{\mathsf{ext}}(\lambda)$ and $\epsilon_{14-\mathsf{col}}(\lambda)$ are negligible.

### D.2 Proof of Theorem 8

Throughout this proof, we will refer to $(\mathbb{G}, g, p)$ as the group $\mathbb{G}$ of prime order $p$ generated by $g$ used in ElGamal. VCC uses four NIP in the algorithm ProveGame. Since these proofs are assumed to be zero-knowledge, there exists four respective simulators $\text{Sim}_0$, $\text{Sim}_1$, $\text{Sim}_2$ and $\text{Sim}_3$ that simulates valid proofs from any statement for the NIP used to produce $\rho_0$, $\rho_1$, $\rho_{2,\alpha}$, and $\rho_{3,\alpha}$ where $\alpha \in [\![n]\!]^*$. We build the following cryptid simulator:

$\text{Sim}_c(\text{set}, t, \text{map}, \alpha^*, \text{clue}_{\alpha^*}, \{(\alpha_i, A_i, j_i)\}_{i \in [\![t]\!]^*})$**:** runs $(\mathsf{pc}_{\alpha^*}, \mathsf{sc}_{\alpha^*}) \leftarrow \mathsf{GenClue}(\text{set}, \text{clue}_{\alpha^*})$ and parses $\mathsf{pc}_{\alpha^*}$ as $(\mathsf{pk}_{\alpha^*}, E_{\alpha^*,1}, E_{\alpha^*,2}, E_{\alpha^*,3})$. For each $\alpha \in [\![n]\!]^* \backslash \{\alpha^*\}$, it picks $(\mathsf{pk}_\alpha, \gamma_{\alpha,1}, \gamma_{\alpha,2}, \gamma_{\alpha,3}) \xleftarrow{\$} \mathbb{G}^4$, then for all $i \in [\![3]\!]^*$ it computes $E_{\alpha,i} \leftarrow \mathsf{Enc}_{\mathsf{pk}_\alpha}(\gamma_{\alpha,i})$. This algorithm sets $\mathsf{pc}_\alpha \leftarrow (\mathsf{pk}_\alpha, E_{\alpha,1}, E_{\alpha,2}, E_{\alpha,3})$. The simulator parses map as $(\mathsf{cell}_i)_{i \in [\![N]\!]^*}$ and each $\text{cell}_i$ as $(T_i, \mathcal{P}_i)$. It picks $\mathsf{pk} \xleftarrow{\$} \mathbb{G}$ and $(\gamma_l)_{l \in [\![14]\!]} \xleftarrow{\$} \mathbb{G}^{15}$, then for all $l \in [\![14]\!]$ it computes $E_l \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\gamma_l)$. For all $i \in [\![N]\!]^*$, it computes $\mathsf{H}(\text{cell}_i) = \prod_{P \in \mathcal{P}_i} P$. This algorithm simulates respectively $\rho_0$ and $\rho_1$ with $\text{Sim}_0$ and $\text{Sim}_1$, and simulates respectively $\rho_{2,\alpha}$

and $\rho_{3,\alpha}$ with $\mathrm{Sim}_2$ and $\mathrm{Sim}_3$ for all $\alpha \in [\![n]\!]^*$, then it sets $\rho \leftarrow (\mathsf{pk}, \rho_0, \rho_1, (\rho_{2,\alpha})_{\alpha\in[\![n]\!]^*}, (\rho_{3,\alpha})_{\alpha\in[\![n]\!]^*})$. According to the rules of Cryptid, the simulator simulates each play $i$ for player $\mathrm{Player}_{\alpha_i}$ (from play 1 to play $t$) by computing $\pi_i \leftarrow \mathrm{Sim}_{A_i}(\mathsf{ElGamal}, \mathsf{pk}_{\alpha_i}, E_{\alpha_i,1}, E_{\alpha_i,2}, E_{\alpha_i,3}, T_{j_i}, \mathcal{P}_{j_i})$. Finally, it returns $\mathsf{view}_{\alpha^*} = (\mathrm{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \rho, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i\in[\![t]\!]^*})$.

Note that this simulator replaces each ciphertext with the encryption of a random element and simulates each NIP .

To prove this theorem, we use the following sequence of games [21] where the first game matches the case where the distinguisher receives elements generated by the real protocol (case $b = 0$ in the confidentiality experiment), and where the last game matches the case where the distinguisher receives elements generated by the simulator (case $b = 1$ in the confidentiality experiment). By showing that each of the intermediate games are computationally indistinguishable from the other, we deduce that the two cases are indistinguishable. We note that the Games $G_0$ and $G_1$ are similar as in the Proof of Theorem 4.

**Game $G_0$:** $\mathcal{D}$ receives as input $(\mathrm{set}, t, \mathrm{map}, \alpha^*, \mathrm{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \rho, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i\in[\![t]\!]^*})$ as in the case $b = 0$ (corresponding to the real protocol) in the confidentiality experiment. we have:

$$\Pr[\mathcal{D} \text{ returns 1 in } G_0] = \Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{VCC},\mathcal{D},0}(\lambda)\right].$$

**Game $G_1$:** This game is the same as $G_0$ except that all the proofs $(\pi_i)_{i\in[\![t]\!]^*}, \rho_0, \rho_1, (\rho_{2,\alpha})_{\alpha\in[\![n]\!]^*}, (\rho_{3,\alpha})_{\alpha\in[\![n]\!]^*}$ are replaced by a simulated proof. Since all NIP are zero-knowledge, We have:

$$\Pr[\mathcal{D} \text{ returns 1 in } G_0] = \Pr[\mathcal{D} \text{ returns 1 in } G_1].$$

**Game $G_2$:** This game is the same as $G_1$ except that each encryption of a clue is replaced by an encryption of a random element. Let $\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda)$ be the IND-CPA advantage of the encryption scheme ElGamal.
We have already shown in the Proof of Theorem 4 that:

$$3(n-1)\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) \geq |\Pr[\mathcal{D} \text{ returns 1 in } G_2] - \Pr[\mathcal{D} \text{ returns 1 in } G_1]|$$

**Game $G_3$:** $\mathcal{D}$ receives as input $(\mathrm{set}, t, \mathrm{map}, \alpha^*, \mathrm{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha\in[\![n]\!]^*}, \rho, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i\in[\![t]\!]^*})$ as in the case $b = 1$ (corresponding to the simulator) in the confidentiality experiment, we have:

$$\Pr[\mathcal{D} \text{ returns 1 in } G_3] = \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{VCC},\mathcal{D},1}(\lambda)\right].$$

To prove indistinguishability between $G_2$ and $G_3$, we use an hybrid argument [14]. We consider a sequence of games from $G_2$ to $G_3$ in which we progressively replace each ciphertext in $G_2$ with the encryption of a random element.

**Game $G_{2,l}$ (for all $l \in [\![14]\!]$):** We define $G_{2,-1}$ as $G_2$ and $G_{2,14}$ as $G_3$. The game $G_{2,l}$ is the same as $G_{2,l-1}$ except that it computes $\gamma_l \xleftarrow{\$} \mathbb{G}$ and sets $E_l \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\gamma_l)$.

We claim that, for any $l \in [\![14]\!]$:

$$\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) \geq |\Pr[\mathcal{D} \text{ returns 1 in } G_{2,l-1}] - \Pr[\mathcal{D} \text{ returns 1 in } G_{2,l}]|$$

We prove this claim by reduction. We build the following p.p.t. algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against the IND-CPA experiment on ElGamal:

$\mathcal{A}_1(\mathsf{pk})$**:** parses $\mathsf{cell}_j$ as $(T_j, \mathcal{P}_j)$, $T_j$ as $V_0$ and $\mathcal{P}_j$ as $(V_l)_{l \in [\![14]\!]}$ where for all $l \in [\![14]\!]^*$, if $\bar{P}_l \in \mathcal{P}_j$, then $V_l = \bar{P}_l$, else $V_l = 1_{\mathbb{G}}$. It sets $m_0 = V_l$ and picks $m_1 \xleftarrow{\$} \mathbb{G}$, and returns $(m_0, m_1)$.

$\mathcal{A}_2(c)$**:** sets $E_l = c$, generates a tuple $\mathsf{input} = (\mathrm{set}, t, \mathrm{map}, \alpha^*, \mathrm{clue}_{\alpha^*}, \mathsf{sc}_{\alpha^*}, (\mathsf{pc}_\alpha)_{\alpha \in [\![n]\!]^*}, \rho, \{(\alpha_i, A_i, j_i, \pi_i)\}_{i \in [\![t]\!]^*})$ as in $G_{2,l-1}$ except that it uses its own values $\mathsf{pk}$ and $E_l$ , and runs $b' \leftarrow \mathcal{D}(\mathsf{input})$. It returns $b'$.

We have:

$$\Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},0}(\lambda)\right] = \Pr[\mathcal{D} \text{ returns 1 in } G_{2,l-1}].$$

$$\Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},1}(\lambda)\right] = \Pr[\mathcal{D} \text{ returns 1 in } G_{2,l}].$$

Thus, we deduce:

$$\begin{aligned}\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) &\geq \left|\Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},0}(\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathsf{ElGamal},\mathcal{A},1}(\lambda)\right]\right| \\ &= |\Pr[\mathcal{D} \text{ returns 1 in } G_{2,l-1}] - \Pr[\mathcal{D} \text{ returns 1 in } G_{2,l}]|\,,\end{aligned}$$

which concludes the proof of the claim. We deduce that:

$$15\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) \geq |\Pr[\mathcal{D} \text{ returns 1 in } G_2] - \Pr[\mathcal{D} \text{ returns 1 in } G_3]|\,.$$

Finally:

$$\begin{aligned}(3n+12)\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda) &\geq |\Pr[\mathcal{D} \text{ returns 1 in } G_0] - \Pr[\mathcal{D} \text{ returns 1 in } G_3]| \\ &= \left|\Pr\left[0 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{VCC},\mathcal{A},0}(\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{Confidentiality}}_{\mathsf{VCC},\mathcal{A},1}(\lambda)\right]\right|,\end{aligned}$$

which concludes the proof of the theorem, since $(3n+12)\epsilon_{\mathsf{IND\text{-}CPA}}(\lambda)$ is negligible.

## E Benchmarks of VCC

We have extended our Rust implementation of CC2 by adding the algorithms ProveGame and VerifyGame in order to analyse the practicality of VCC. The source code for our implementation is available at `https://anonymous.4open.`

`science/r/cryptidscheme`. The table 3 provides an overview of an average computation time, in seconds, of the algorithms ProveGame and VerifyGame of the scheme VCC, and the table 4 describes an average size, in kilobytes, of the proof of the correct game. Our measurements are based on 500 runs and depend on the number of players: 3, 4, and 5. Again, we did not evaluate efficiency on real games, but on randomly generated maps and clues. Since clues of the form "one of two types" are less common than clues of the form "so many cells away from something", we decide to use $\lfloor n/2 \rfloor$ clues of the first type and $\lceil n/2 \rceil$ clues of the second type, where $n$ is the number of players.

The proof that the game is correct takes less than 5 seconds and must be done for the 180 game setups by the game editor. It will therefore take 15 minutes for the whole Cryptid game, bearing in mind that this operation is only performed once, a priori, before the game distribution. The verification of the game takes less than 5 seconds. This operation must be performed by each player. The execution time of this operation is a little long, but we remind the reader that it is only done once before the start of the game.

The size of the proof of each game setup is less than 1300 kilobytes, knowing that these proofs must be downloaded at the same time as the public clue keys when the game is acquired. Downloading the full game will therefore require less than $9360 + 180 \times 1300 = 243360$ kilobytes of storage.

**Table 3.** Running time in seconds for our verifiable cryptographic cryptid scheme. The results are an average over 500 executions.

| Scheme | VCC | | |
|---|---|---|---|
| Number of player | 3 | 4 | 5 |
| ProveGame | 4.07 | 4.70 | 5.31 |
| VerifyGame | 3.11 | 3.56 | 4.02 |

**Table 4.** Size in kilobytes for our verifiable cryptographic cryptid scheme. The results are an average over 500 executions.

| Scheme | VCC | | |
|---|---|---|---|
| Number of player | 3 | 4 | 5 |
| $\rho$ | 975.44 | 1116.02 | 1256.32 |