GAIT: A Game-Theoretic Defense Against Intellectual Property Theft

Youzhi Zhang, Dongkai Chen, Sushil Jajodia, *Life Fellow, IEEE*, Andrea Pugliese, *Member, IEEE*, V. S. Subrahmanian, and Yanhai Xiong

***Abstract*—Months may pass before the victim of IP theft even knows they have been compromised. During this time, the attacker can exfiltrate large amounts of data. Recent work has proposed the idea of injecting a set of believable fake versions of a real document into a network so that the attacker has to expend time and effort to identify the real document from a sea of similar documents. In this paper, we consider the problem of an attacker who is smart and breaks a technical document down into small, bit-sized "units" and inspects them one by one so as to defeat the fake document defense. If a unit in a document is determined to be fake, the adversary does not need to look further at the same document. He can also immediately identify as fake, any other document that contains the same unit. In this paper, we consider the problem of a smart attacker using this strategy. Our proposed defensive algorithm, called GAIT, is shown to be successful in mitigating such attacks. GAIT can work in conjunction with any NLP-based generative method to create fake technical documents.**

***Index Terms*—H.2.0.a security, integrity, and protection < H.2.0 general < H.2 database management < H information technology and systems.**

I. Introduction

IP THEFT is a growing problem. A recent article [1] alleges that trillions of dollars of IP were stolen via cyberattacks in recent years. The FBI has an entire division focused on cyber-enabled intellectual property theft. Yet, successful attacks continue. On average, 312 days may elapse between the time a victim organization is first compromised by a zero day attack and the time when the attack is discovered [2]. This means the adversary has almost a year to steal intellectual property from the victim organization.

One solution is to generate n believable fake versions of any given *technical* document. In this paper, we do not restrict the generative method: it could be done using deep neural network based systems like GPT2 or GPT3 or more traditional methods to generate language such as FORGE [3], [4]. An IP thief is then confronted with $(n+1)$ documents, only one of which is real. If the documents are believable, he may end up believing a fake document is the real thing. Even if he discovers the real document, a considerable amount of time, money, and frustration is expended by the attacker.[1] Several recent works have looked at the generation of fakes [3], [4], [5], [6].

The Problem and Approach: No past work on fake document generation has studied how the adversary may identify the fake documents. His goal is to identify the real document as fast as possible. Our goal is to impose as much cost on the adversary (e.g., delay him as much as possible, maximize mistaken identification of the real document), *given his approach to finding the real document*. This immediately suggests a game-theoretic approach to the problem which is what we use in GAIT. Because past work [3], [4], [5] has already addressed the problem of maximizing mistakes by the adversary by generating text intelligently, this paper builds on past work on fake document (text) generation and tries to maximizes the time expended by the adversary.

None of these efforts consider how an adversary may adapt to efforts to deceive him via fake technical documents — and by extension, how we may generate fake documents to minimize the attacker's utility (or maximizing the time the attacker must spend in order to find the real document) by delaying his discovery of the real document as much as possible.

We propose a Stackelberg game based approach to this problem. A defender (victim) splits a document into a given number of units which may or may not be the same size. A unit can be a subsection or section or paragraph or even a sentence. He can then generate fake versions of a document by replacing some units from the original with fakes. An attacker looking at a given document from the $(n+1)$ versions of a document (one real, n fake), may examine a unit in detail. If he determines that it is

Manuscript received 7 December 2022; revised 15 May 2023; accepted 24 July 2023. Date of publication 26 July 2023; date of current version 11 July 2024. This work was supported by ONR under Grants N00014-18-1-2670 and N00014-20-1-2407. *(Corresponding author: V. S. Subrahmanian.)*

Youzhi Zhang is with the Centre for Artificial Intelligence and Robotics, Hong Kong Institute of Science & Innovation, Chinese Academy of Sciences, Hong Kong SAR, China (e-mail: youzhi.zhang@cair-cas.org.hk).

Dongkai Chen is with the Department of Computer Science, Dartmouth College, Hanover, NH 03755 USA (e-mail: dongkai.chen.gr@dartmouth.edu).

Sushil Jajodia is with the Center for Secure Information Systems, George Mason University, Fairfax, VA 22030 USA (e-mail: jajodia@gmu.edu).

Andrea Pugliese is with the University of Calabria, 87036 Rende, Italy (e-mail: andrea.pugliese@unical.it).

V. S. Subrahmanian is with the Department of Computer Science and with the Buffett Institute for Global Affairs, Northwestern University, Evanston, IL 60208 USA (e-mail: vss@northwestern.edu).

Yanhai Xiong is with the Department of Data Science, William & Mary, Williamsburg, VA 23185 USA (e-mail: yanhaixiong7@gmail.com).

Digital Object Identifier 10.1109/TDSC.2023.3299225

[1] Legitimate users need to distinguish the real document from the fake ones. This problem has already been solved by [3] using message authentication codes. If we have n fakes and one real document, each of the $(n+1)$ documents has a public key. A combination of the user's private key and the document's public key can then be used by the user to decide whether a document is real or not. As this problem has been previously solved, we do not discuss it further in this paper and assume it will be used by enterprises using fake documents to deter IP theft.

real, then he can mark the unit to be real in all $n+1$ versions. Likewise, if he decides it is fake, he can mark the unit to be fake in all $(n+1)$ versions — and pruning from future consideration, any document with at least one fake unit. Such an intelligent adversary can significantly speed up his inspection of the $(n+1)$ versions of a real document. From the defender's point of view, he must generate the fake versions of a document in such a way that the attacker has to spend as long as possible in studying the entire document collection, defeating the attacker's pruning strategy mentioned above. We make the worst-case assumption, common in security, that the adversary will correctly determine whether a unit is fake/real if he examines the unit. This is consistent with cases where the IP thieves are backed by nation states who certainly have the sophistication and technical people who can make such determinations. GAIT can also work on top of any NLP-based method used by the attacker to decide if a unit is real or fake.

GAIT models the attacker's decision problem as a Markov Decision Problem. The defender's decision problem is characterized as a Stackelberg game — we show that it can be solved via a probability flow strategy that captures the solution via a Nash equilibrium computation. However, this solution is computationally very expensive (and we show the attacker's reasoning to be NP-hard). We propose a novel GAIT_AO algorithm that incrementally expands the attacker's strategy space. We show that GAIT_AO converges to a Nash equilibrium.

We run experiments showing that the defender using GAIT with Nash Equilibrium will play optimally if the attacker sticks to his Nash equilibrium — but if the attacker deviates from his Nash equilibrium, then the defender will be more successful in deceiving the attacker (i.e., the defender gains utility). We further show that compared to various baselines, our GAIT_AO approach is fast and outperforms many baselines.

The paper is organized as follows. Section II presents related work. We formally define the GAIT model in Section III. We develop the nonlinear and linear programs used to compute the defender's optimal strategy in Section IV. Next, we develop the much more scalable GAIT_AO algorithm in Section V. Our experimental results are shown in Section VI. We close with conclusions and future work in Section VII.

II. Related Work

Cyber deception has long been used to protect IP. "Honey files" (e.g., `password.txt` or `SSN.docx`) have long been used as bait that, when accessed by attackers, trigger security alarms [7]. Another example is the D^3 system [8] that uses attractive file names to deceive attackers.

Fake Version Generation: Another line of research for cyber deception is to generate fake versions of the original IP to fool and deter attackers. [9], [10] generate fake code in order to deceive attackers. [11] uses "Canary Files" to detect unauthorized access. In a subsequent paper [12], the same authors propose seven criteria that fake documents must satisfy. An important criterion is that the fakes must be believable, yet not reveal important information. None of these papers, however, automatically generates fake natural language content of the type present in technical documents.

Fake Text Document Generation: A few studies focus on generating fake technical documents. [13] uses separate modules to generate fake documents for different topics. FORGE [3] and WE-FORGE [4] automatically generate fake versions of any real document using ontologies and a combination of word embeddings and clustering, respectively. FEE [6] focuses on generating fake equations using a framework based on context-free grammars. [14] generates fake knowledge graphs to combat IP theft. Probabilistic logic graphs [5] have been proposed as a single framework within which to handle different modalities (e.g., tables, formulas, equations, text) inside a technical document.

All existing approaches focus on generating multiple believable fakes versions of an original document. *None of them consider how attackers might react strategically to the injection of fake documents. In this paper, we study how attackers would target a fake document generation system (e.g. [3], [4], [5]) so that they can more easily separate the real document from the fakes.* The GAIT framework proposed in this paper shows one way in which attackers can reduce the benefits of *any* fake document generation system, as well as a way in which we can defend against such attacks.

Finally, recent efforts for generating fake text such as ChatGPT differ from the work in this paper in the following respects. First, they typically do not try to generate a fake version of a given document. Rather they generate text consistent with a given distribution and request. Second, systems such as [3], [4] generate n fake versions of a technical document, not one. This requires ensuring that the fakes are different from the real document and from each other (so the fakes do not all look the same). Third, systems such as [3], [4] focus on technical documents, while systems such as ChatGPT do not. This does not mean ChatGPT will not be useful in the future, but these capabilities have not been shown to date.

Stackelberg Games for Security: Systems based on Stackelberg security games have been widely used to improve cybersecurity and other forms of security [15]. There are also several excellent surveys of deception and game theoretic methods for cybersecurity [16], [17], [18], [19].

In many security games, the attacker and defender have the same action space, i.e., they are either attacking or defending the same target, e.g., terminals in an airport [20]. However, in the situation we study in this paper, the action spaces of the defender and the attacker are different. There is some past work where different action spaces for the attacker and defender are considered. In network security games [21], [22], [23], for example, the defender tries to interdict the attacker who is either escaping via a path or following a path to attack a target. However, in the fake document problem considered in this paper, the attacker's strategy cannot be modeled as a path over nodes in a network.

Incremental strategy generation algorithms (double oracle, single oracle) [21], [23], [24] have been previously used for computing equilibria. In dynamic games, an incremental strategy generation algorithm using a branch-and-bound best response

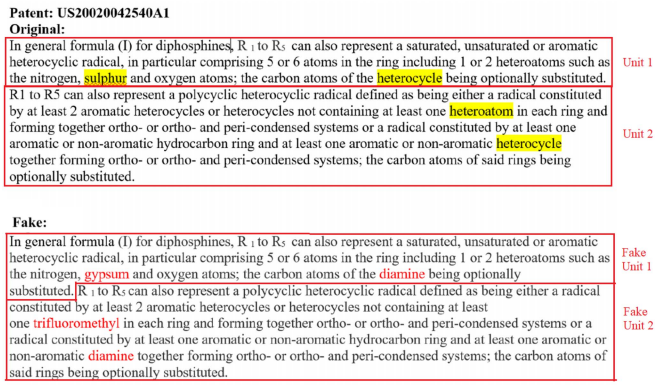


Fig. 1. Two units of a document from US Patent US20020042540A1 [27]. The top part is the original content, with text highlighted with yellow to be changed. The bottom part is the fake document generated with [4], with red context replacing the original words.

oracle was first used in [24] for general games, and was later extended to specific games [23]. However, such algorithms cannot effectively estimate upper bounds of actions and utilities of temporary terminal states in our problem. Hence, new game theoretic techniques are needed.

III. GAIT's Game-Theoretic Formulation

We now describe how GAIT formulates the fake version generation problem. Consider an inventor who is writing up an invention. We envisage a world in which a system such as GAIT automatically generates fake versions of this document.

In this paper, we are not concerned with the linguistic aspects of the fake document generation process. *Our proposed framework is independent of the specific NLP-based method* (including, e.g., GPT-3 [25], BERT [26], FORGE [3], and WE-FORGE [4]) used to generate fake documents.

The question we consider is how an adversary might defeat such an algorithm and quickly discover the real document from a sea of fakes. To do this, we assume that the document has M "units" in all and that we want to generate n fake documents. What constitutes a unit can be decided by the author of the document — it is not necessary that units be homogeneous. For instance, the first unit may be a paragraph, the second may be a subsection, and so forth. The author designates what the units are in his document. In this paper, we only consider units consisting of text. Real-world documents may contain a mix of text, tables, equations, and diagrams. [5] have introduced probabilistic logic graphs (PLGs) to provide a uniform representation of multimodal document content to generate fake multimodal documents. The generation of a game-theoretic approach for fake multimodal document generation is left for future work.

We represent the real and fake versions as a matrix D of size $(n+1) \times M$. Each row in this matrix corresponds to a document and each column to a single unit. With a slight abuse of notation, we use D to also represent the set of documents $\{D_1, \ldots, D_{n+1}\}$. We assume unit j has a set F_j of fake versions.

Example 1: Fig. 1 shows a part of a US patent. This part is divided into two units ($M = 2$). There are $|F_1|$ and $|F_2|$ fake versions for them, respectively. The "Fake" part shows one fake document with a fake version for each unit.

The top left matrix in Fig. 2 shows a matrix D representing 5 documents with 5 units each — the third row corresponds to the real document.

Function $real(D_{i,j})$ returns 1 if $D_{i,j}$ is real and 0 otherwise. Thus, there is exactly one row i in D such that $real(D_{i,j}) = 1$ for all $1 \leq j \leq M$ and this row corresponds to the real document.[2]

Example 2: In the case of Example 1, assume that (i) the original (resp., fake) document in Fig. 1 is D_1 (resp., D_2), (ii) $|F_1| = |F_2| = 1$, and (iii) documents D_3 and D_4 contain exactly one fake unit each. Then, we have:

- $real(D_{1,1}) = 1, real(D_{1,2}) = 1,$
- $real(D_{2,1}) = 0, real(D_{2,2}) = 0,$
- $real(D_{3,1}) = 0, real(D_{3,2}) = 1,$
- $real(D_{4,1}) = 1, real(D_{4,2}) = 0.$

Our framework is independent of which NLP-based method is used to determine whether a unit is real or fake. The attacker can use any such fake document detection method, or even a two-stage method where an automated fake document generator is used to flag documents as potentially fake – and then in a second stage, a human might review such a method. *We make the worst case assumption that if the attacker examines a unit then he will always be able to correctly determine if it is real or not* — in other words, we make the common assumption in cybersecurity that the attacker is extremely smart.

We may start building D by deciding that the i'th row corresponds to the real document. For each other entry $D_{i',j}$ in each column j, with $i' \neq i$, we need to choose one of the $|F_j| + 1$ versions (one real, $|F_j|$ fake) of the real unit $D_{i,j}$. Going back to Fig. 2, after putting the real document in row 3 and the real unit in $D_{1,1}$, we would need to decide what to place in cell $D_{2,1}$. Should we put the real unit $D_{1,1}$ or a fake version of it drawn from the set F_1 of fakes?

Thus, the defender is choosing n fake documents from a set of $N_f = (\Pi_{1 \leq j \leq M}(|F_j| + 1)) - 1$ fakes, i.e., he has to draw one selection from $\binom{N_f}{n}$ options.

All mathematical notations used throughout the paper are summarized in Table I.

A. Defender's Strategy Space

The fake document generation problem for the defender is about how to generate a set of n fake documents to combat IP theft. Given a set of fake versions of each unit, the defender needs to determine what matrix D should be.

We can think of the matrix D as a blank matrix except for one row filled in with the units of the real document. An "instance" of D is a way of filling in D's other entries. In game theoretic terms, the *pure strategy space* $\mathcal{D}$ consists of all possible instances of D. A *mixed strategy* is a probability distribution over $\mathcal{D}$, denoted by x, where $x(D)$ represents the probability of using a specific instance D. We use X to denote the mixed strategy space.

[2]The defender knows the function $real(D_{i,j})$. As is common in cybersecurity, we make the worst case assumption that the attacker is very smart and can accurately determine whether a unit is fake or not if he examines it.

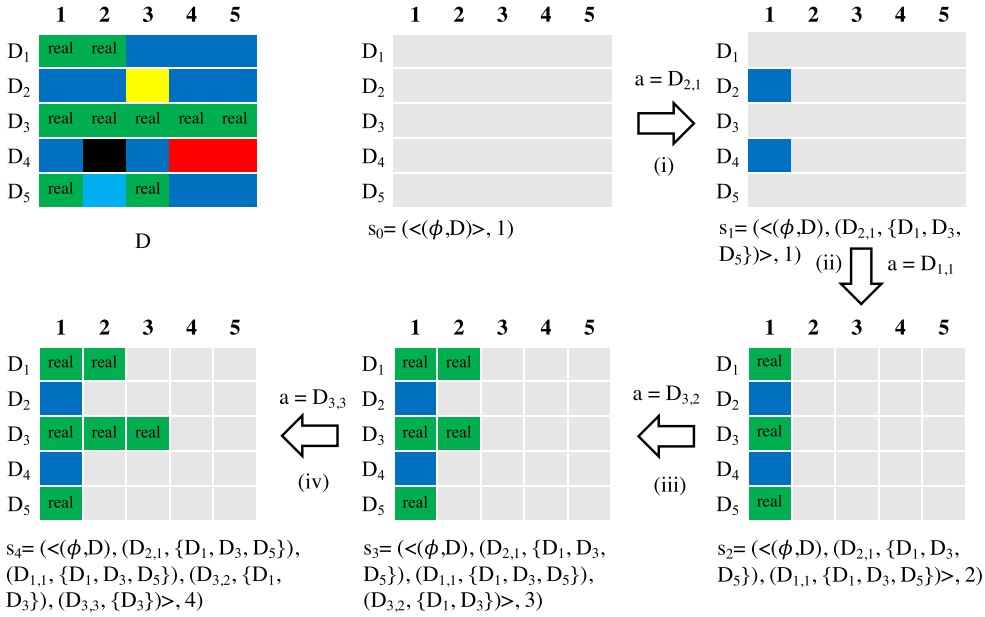


Fig. 2. $D = \{D_1, \ldots, D_5\}$, where D_3 is the real document, is a pure strategy of the defender. Each color in each column represent a version of a unit, and green represents the real version. At the initial state s_0, any document in D could be real. (i) After examining $D_{2,1}$, the attacker knows that $D_{2,1}$ and $D_{4,1}$ are the same version and fake, so the real document is in $\{D_1, D_3, D_5\}$ — the attacker reaches state s_1. (ii) After examining $D_{1,1}$, the attacker knows that the first unit of the documents in $\{D_1, D_3, D_5\}$ is real, so the real document is still in $\{D_1, D_3, D_5\}$ — the attacker reaches s_2. (iii) After examining $D_{3,2}$, the attacker knows that the second unit of the documents in $\{D_1, D_3\}$ is real, so the real document is in $\{D_1, D_3\}$ — the attacker reaches s_3. (iv) After examining $D_{3,3}$, the attacker knows that the third unit of D_3 is real, so the real document is in $\{D_3\}$ — the attacker reaches s_4 and concludes that D_3 is real.

We use $first\text{-}real(D, j) = \{D_i \in D \mid real(D_{i,j'}) = 1, 1 \leq j' \leq j)\}$ to denote the set of documents in a given instance D whose first j units are real. That is, if the attacker has only looked at the first j units, then he cannot rule out any of the documents in $first\text{-}real(D, j)$. For convenience, we set $first\text{-}real(D, 0) = D$, i.e., any document in D could be real before the attacker starts his examination of the units.

B. Intuition Behind Attacker's Method

In this section, we provide examples to illustrate how an attacker might uncover the real document.

Example 3: In the case of Example 1:

i) Suppose the attacker looks at $D_{2,1}$ which is the first unit of document D_2. In this case, he determines that $D_{2,1}$ is fake. He infers that $D_{3,1}$ is fake because it is identical to $D_{2,1}$. Hence, he knows D_2, D_3 are both fake.
ii) The attacker looks at $D_{1,1}$. He realizes it is real. He cannot prune anything because $D_{1,1} = D_{4,1}$. Thus, $first\text{-}real(D, 1) = \{D_1, D_4\}$ as these two documents are still plausible.
iii) The attacker now moves on to the second unit. He looks at $D_{1,2}$ and realizes it is real and moreover $D_{1,2} \neq D_{4,2}$ — so D_4 is fake and $first\text{-}real(D, 2) = \{D_1\}$. Thus, the real document is D_1 and the attacker is done.

Example 4: Consider the sample D shown at the top left of Fig. 2. Cells colored with the same color in each column have the same textual content.

i) Suppose the attacker looks at $D_{2,1}$ which is the first unit of document D_2. In this case, he determines that $D_{2,1}$ is fake. He infers that $D_{4,1}$ is fake because it is identical to $D_{2,1}$. Hence, he knows D_2, D_4 are both fake.
ii) The attacker looks at $D_{1,1}$. He realizes it is real. He cannot prune anything because $D_{1,1} = D_{3,1} = D_{5,1}$. Thus, $first\text{-}real(D, 1) = \{D_1, D_3, D_5\}$ as these three documents are still plausible.
iii) The attacker now moves on to the second unit. He first looks at $D_{3,2}$ and realizes it is real – and as $D_{1,2} = D_{3,2}$ but $D_{1,2} \neq D_{5,2}$, it follows that $first\text{-}real(D, 2) = \{D_1, D_3\}$.
iv) The attacker looks at $D_{3,3}$. He sees that this unit is real and moreover $D_{1,3} \neq D_{3,3}$ — so D_1 is fake and $first\text{-}real(D, 3) = \{D_3\}$. Thus, the real document is D_3 and the attacker is done.

Fig. 3 shows another way for the attacker to discover the real document — in this case, the attacker has to examine 5 units instead of 4.

C. Attacker's Strategy Space

Given an instance D generated by the defender, we have illustrated how the attacker reasons about D in order to uncover the real document. We assume the attacker:

1) knows D — this is reasonable as the attacker knows what documents he has exfiltrated from a compromised network;

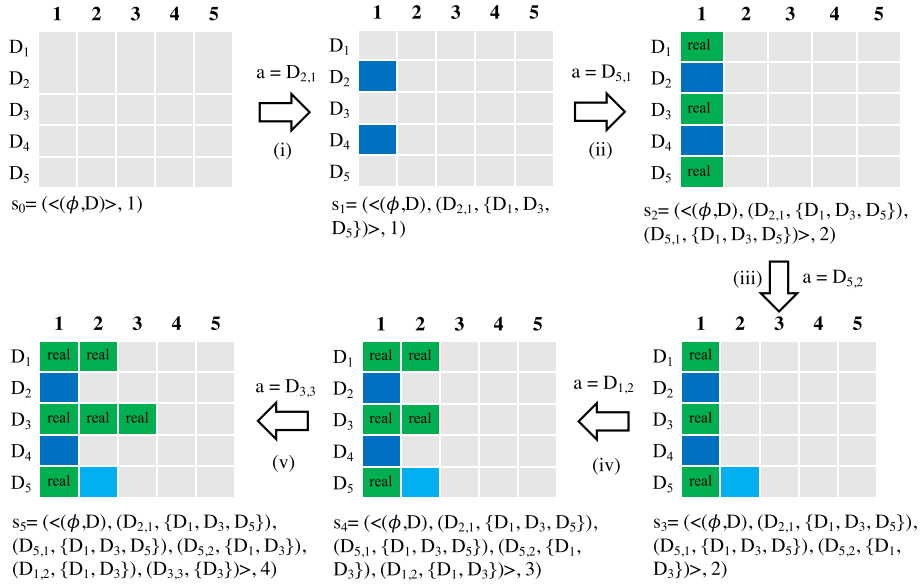


Fig. 3. Another way for the attacker to find the real document starting from matrix D in Fig. 2. (i) After examining $D_{2,1}$, the attacker knows that $D_{2,1}$ and $D_{4,1}$ are the same version and fake, so the real document is in $\{D_1, D_3, D_5\}$ — the attacker reaches state s_1. (ii) The attacker now examines $D_{5,1}$ and discovers it is real, so the real document is still in $\{D_1, D_3, D_5\}$ — the attacker reaches s_2. (iii) After examining $D_{5,2}$, the attacker knows that $D_{5,2}$ is fake, so the real document is in $\{D_1, D_3\}$ — the attacker reaches s_3. (iv) After examining $D_{1,2}$, the attacker knows that the second unit of documents in $\{D_1, D_3\}$ is real, and reaches s_4. (v). After examining $D_{3,3}$, the attacker knows it is real, so the real document is in $\{D_3\}$ — the attacker reaches s_5 and concludes that D_3 is real.

2) knows the number of units M — we make the worst case assumption commonly made in cybersecurity as the attacker may try many different possible values of M and get to know M.

We assume additionally that it costs the attacker $t(D_{i,j})$ units of time to examine unit j in document i in order to determine whether it is real or fake.[3] The problem for the attacker is to identify, as quickly as possible, which row in D represents the original document.

As seen above, at each unit/step/column, the attacker must decide which document to examine. After checking a unit of a document, the attacker can update his inferences as discussed earlier. For instance, if the attacker examines $D_{i,j}$ and knows that $real(D_{i,j}) = 1$, he can infer that $real(D_{i',j}) = 1$ for all i' such that $D_{i,j} = D_{i',j}$. Likewise, if he knows $real(D_{i,j}) = 0$, he can infer that $real(D_{i',j}) = 0$ for all i' such that $D_{i,j} = D_{i',j}$. Before moving to the next unit, the attacker has to find the real version for the current unit. When trying to find the real version of the j-th unit, the attacker may look at the j-th unit of a specific document. If the version of this unit is real, he can immediately identify the set $first\text{-}real(D, j)$ of all documents in D whose first j units are real.

Histories: As shown above, as the attacker looks at units, he can iteratively eliminate documents that cannot possibly be real. We introduce the concept of histories below and show how an attacker's choice of units to examine will change the sets of documents still considered potentially real.

Suppose the attacker has already computed $first\text{-}real(D, j-1)$ and is considering $D_{i,j}$. In this case, he has observed documents in a set $obs\text{-}real(D, j)$ such that $first\text{-}real(D, j) \subseteq obs\text{-}real(D, j) \subseteq first\text{-}real(D, j-1)$. We call $(D_{i,j}, obs\text{-}real(D, j))$ an *observed real document set pair*. We can think of such a pair to be an action (by the attacker to examine $D_{i,j}$) and the observations of what is real based on this choice. An *observed real document set history* $h = \langle(\phi, D), \ldots, (D_{i,j}, obs\text{-}real(D, j))\rangle$ is a sequence of observed real document set pairs starting from the initial pair with the whole set of documents D and the empty action ϕ. We use the following notation:

- $I(D_{i,j}) = D_i$, i.e., it returns the document to which $D_{i,j}$ refers – we assume $I(\phi) = 0 \notin D$;
- $J(D_{i,j}) = j$, i.e., it returns the index of the unit to which $D_{i,j}$ refers – we assume $J(\phi) = 1$;
- $common\text{-}unit(D, i, j)$ is the set of documents in D whose j-th unit is equal to $D_{i,j}$;
- $last\text{-}set(h)$ returns the observed real document set in the last pair in history h;
- $last\text{-}unit(h)$ returns the examined unit in the last pair in history h.

We can now define the set H_D of histories generated by D recursively. We start with $h_0 = \langle(\phi, D)\rangle \in H_D$. This says that initially, the empty action is in place and all documents in D are possibly real. Then, for each $h \in H_D$:

1) If $|last\text{-}set(h)| > 1$ and $I(last\text{-}unit(h)) \in last\text{-}set(h)$, then $h \cup \langle(D_{i,j}, obs\text{-}real(D, j))\rangle \in H_D$ with $D_i \in last\text{-}set(h)$, $j = J(last\text{-}unit(h)) + 1$, and $obs\text{-}real(D, j)) \subseteq last\text{-}set(h)$ is computed as follows:
   a) $obs\text{-}real(D, j) = first\text{-}real(D, j)$ if $real(D_{i,j}) = 1$;
   b) $obs\text{-}real(D, j) = last\text{-}set(h) \setminus common\text{-}unit(D, i, j)$ if $real(D_{i,j}) = 0$.

[3] GAIT imposes no restriction on how $t(D_{i,j})$ should be set. For instance, the deceiver may believe the attacker will take more time if $D_{i,j}$ is being manually inspected and less time if it is fully automated using a fake document detection tool. In this case, he can set $t(D_{i,j})$ to be commensurate with these expectations.

TABLE I
SYMBOLS USED

n	Number of fake documents
M	Number of units in a document
D	Document-unit matrix (or set of documents)
D_i	i-th document in D
$D_{i,j}$	j-th unit in D_i
F_j	Set of fake versions for unit j
$real(D_{i,j})$	1 if $D_{i,j}$ is real, 0 otherwise
N_f	Number of possible fake documents
$\mathcal{D}$	Pure defender strategy (all possible instances of D)
x	Mixed defender strategy (probability distribution over $\mathcal{D}$)
X	Mixed defender strategies (all possible functions x)
$t(D_{i,j})$	Time to compute $real(D_{i,j})$
$first\text{-}real(D, j)$	Set of documents in D whose first j units are real
$obs\text{-}real(D, j)$	Set of observed real documents at the j-th unit
$(D_{i,j}, obs\text{-}real(D, j))$	Observed real document set pair
h	Observed real document set history (sequence of pairs)
$I(D_{i,j})$	D_i (document to which $D_{i,j}$ refers)
$J(D_{i,j})$	j (index of the unit represented by $D_{i,j}$)
$common\text{-}unit(D, i, j)$	Set of documents in D whose j-th unit is equal to $D_{i,j}$
$last\text{-}set(h)$	Observed real document set in the last pair in h
$last\text{-}unit(h)$	Examined unit in the last pair in h
H_D	Set of histories generated by D
$\mathcal{H}$	All possible histories H_D
S	Set of states
s	State
h_s	Observed real document set history in state s
j_s	Column index of the unit to be examined in state s
s_0	Initial state
s_t	Terminal state
S_t	Set of terminal states
A_s	Action set in state s
A	Action set in S
a	Action (unit to be examined)
$S_{s,a}$	Set of states possibly reached from s by taking action a
π	Pure attacker strategy
y	Behavior attacker strategy
Y	Behavior attacker strategies (all possible functions y)
u_a	Utility function of the attacker
u_d	Utility function of the defender
$P_y(s)$	Probability to reach state s under y
U_a	Expected utility of the attacker
U_d	Expected utility of the defender
$f(s, a)$	Probability that the attacker reaches state s and takes action a
$V(s)$	Value of state s
$B(a)$	Upper bound for action a
$B(a, s')$	Upper bound for succeeding state s' of action a
$b(s)$	Lower bound for state s
$V(a)$	Value of action a
$B(D, j)$	Minimum time to reach a terminal state starting from the j-th column
$B(a, D', D)$	Minimum time to reach a terminal state by taking action a under $first\text{-}real(D, J(a)) \subseteq D' \subseteq D$ with $\lvert D'\rvert > 1$ and $I(a) \in D'$

2) If $|last\text{-}set(h)| > 1$ and $I(last\text{-}unit(h)) \notin last\text{-}set(h)$, then $h \cup \langle (D_{i,j}, obs\text{-}real(D, j)) \rangle \in H_D$ with $D_i \in last\text{-}set(h)$, $j = J(last\text{-}unit(h))$, and $obs\text{-}real(D, j) \subseteq last\text{-}set(h)$ computed as follows:
   a) $obs\text{-}real(D, j) = first\text{-}real(D, j)$ if $real(D_{i,j}) = 1$;
   b) $obs\text{-}real(D, j) = last\text{-}set(h) \setminus common\text{-}unit(D, i, j)$ if $real(D_{i,j}) = 0$.

Note that there is no succeeding history to h with $|last\text{-}set(h)| = 1$. We use $\mathcal{H}$ to denote the set $\cup_{D \in \mathcal{D}} H_D$.

Example 5: In the case of Fig. 2, the history h is initially $\langle (\phi, D) \rangle$ and therefore $last\text{-}set(h) = D$. Then:

- In transition (i), the attacker examines unit $D_{2,1}$, which is the first to be examined on its column. We have $|last\text{-}set(h)| = |D| > 1$, $I(last\text{-}unit(h)) = 0 \notin last\text{-}set(h)$, and $real(D_{2,1}) = 0$. Thus, according to Case 2b of the above definition of the set of histories, the pair $(D_{2,1}, \{D_1, D_3, D_5\})$ is added to the history. As a consequence, $last\text{-}unit(h) = D_{2,1}$ and $last\text{-}set(h) = \{D_1, D_3, D_5\}$.
- In transition (ii), the attacker examines unit $D_{1,1}$, which is the second to be examined on its column. We have $|last\text{-}set(h)| = |\{D_1, D_3, D_5\}| > 1$, $I(last\text{-}unit(h)) = D_2 \notin last\text{-}set(h)$, and $real(D_{1,1}) = 1$. Thus, according to Case 2a of the definition, the pair $(D_{1,1}, \{D_1, D_3, D_5\})$ is added to the history. As a consequence, $last\text{-}unit(h) = D_{1,1}$ and $last\text{-}set(h) = \{D_1, D_3, D_5\}$.
- In transition (iii), the attacker examines unit $D_{3,2}$, which is the first to be examined on its column. We have $|last\text{-}set(h)| = |\{D_1, D_3, D_5\}| > 1$, $I(last\text{-}unit(h)) = D_1 \in last\text{-}set(h)$, and $real(D_{3,2}) = 1$. Thus, according to Case 1a of the definition, the pair $(D_{3,2}, \{D_1, D_3\})$ is added to the history. As a consequence, $last\text{-}unit(h) = D_{3,2}$ and $last\text{-}set(h) = \{D_1, D_3\}$.
- In transition (iv), the attacker examines unit $D_{3,3}$, which is the first to be examined on its column. We have $|last\text{-}set(h)| = |\{D_1, D_3\}| > 1$, $I(last\text{-}unit(h)) = D_3 \in last\text{-}set(h)$, and $real(D_{3,2}) = 1$. Thus, according to Case 1a of the definition, the pair $(D_{3,3}, \{D_3\})$ is added to the history. As a consequence, $last\text{-}unit(h) = D_{3,3}$ and $last\text{-}set(h) = \{D_3\}$.
- The history is now $h = \langle (\phi, D), (D_{2,1}, \{D_1, D_3, D_5\}), (D_{1,1}, \{D_1, D_3, D_5\}), (D_{3,2}, \{D_1, D_3\}), (D_{3,3}, \{D_3\}) \rangle$. Since $|last\text{-}set(h)| = 1$, no other units are examined.

Attacker's Markov Decision Process: Formally, the attacker's decision problem is modeled as a Markov Decision Process (MDP) with a set S of states. Each state $s = (h_s, j_s) \in S$ consists of the observed real document set history h and the index j_s of the unit to be examined. The initial state is $s_0 = (h_0, 1)$ and the terminal state is $s_t = (h_{s_t}, j_{s_t})$ with $|last\text{-}set(h_{s_t})| = 1$ according to our assumption that there is only one real document in D. The set of terminal states is S_t. An action is a unit to be examined. For each non-terminal state s, the action set is $A_s = \{D_{i,j_s} \mid D_i \in last\text{-}set(h_s)\}$, i.e., the attacker only needs to check the j_s-th unit of documents in $obs\text{-}real(D, j_s)$ at s. The whole action set is $A = \cup_{s \in S} A_s$.

Suppose $S_{s,a}$ denotes the set of states possibly reached from s by taking action $a \in A_s$. It is defined as $S_{s,a} = S_{s,a}^{\text{real}} \cup S_{s,a}^{\text{fake}}$ where:

- $S_{s,a}^{\text{real}} = \{s' \mid s' = (h_s \cup \langle (a, obs\text{-}real(D, j_s)) \rangle, j_s + 1), \forall obs\text{-}real(D, j_s) \subseteq last\text{-}set(h_s), \exists D \in \mathcal{D}, h_s \in H_D, obs\text{-}real(D, j_s) = first\text{-}real(D, j_s), h_s \cup (a, obs\text{-}real(D, j_s)) \in H_D, real(a) = 1\}$, i.e.,

if a is real, then the attacker will move to the next unit while updating the observed real document set at the j_s-th unit. To make s transition to s' in this case, there exists D such that D and s have the same observed real document set history up to j_s, and D's real document set at j_s is equal to $obs\text{-}real(D, j_s)$ including $I(a)$. The state transitions (ii) and (iii) in Fig. 2 are two examples for this case. The state transition (iv) in Fig. 2 is also an example for this case, which transits to a terminal state.

- $S^{\text{fake}}_{s,a} = \{s' \mid s' = (h_s \cup \langle (a, obs\text{-}real(D, j_s))\rangle, \ j_s), \forall obs\text{-}real(D, j_s) \subseteq last\text{-}set(h_s), \ \exists D \in \mathcal{D}, h_s \in H_D, \ first\text{-}real(D, j_s) \subseteq obs\text{-}real(D, j_s), h_s \cup \langle (a, obs\text{-}real(D, j_s))\rangle \in H_D, \ common\text{-}unit(D, i, j) \cup obs\text{-}real(D, j_s) = last\text{-}set(h_s)$ (with $a = D_{i,j}$), $real(a) = 0, I(a) \notin obs\text{-}real(D, j_s)\}$, i.e., if a is fake, then the attacker will update the observed real document set at the j_s-th unit and stay at the j_s-th unit in order to try to find the true one. To make s transit to s' in this case, there exists D such that D and s have the same observed real document set history, and D's observed real document set at j_s is a subset of $obs\text{-}real(D, j_s)$. In addition, $last\text{-}set(h_s)$ should be equal to the union of $obs\text{-}real(D, j_s)$ and $common\text{-}unit(D, i, j)$. The state transition (i) in Fig. 2 is an example for this case.

We say the states in $S_{s,a}$ are succeeding states of s.

Attacker's Pure and Behavior Strategies: A pure strategy for the attacker is a deterministic policy mapping each non-terminal state to an action, i.e., $\pi : S \setminus S_t \to A$. A behavior strategy of the attacker $y : (S \setminus S_t) \times A \to [0, 1]$ is a probability distribution over A_s in each non-terminal state s, i.e., $y(s, a)$ is the probability of taking action $a \in A_s$ in state s. We use Y to denote the behavior strategy space.

D. Utilities

We now come to the problem of defining the utility functions. We assume that the game is zero-sum because the defender tries to maximize the time the attacker must spend before finding the real document, while the attacker tries to minimize this time.

Attacker's Utility: We therefore define the attacker's utility function $u_a : S \to \mathbb{R}$ as $u_a(s) = -\sum_{(D_{i,j},\cdot)\in h_s} t(D_{i,j})$, that is the negative of the sum of the times expended in examining documents.

Defender's Utility: The defender's utility is $u_d(s) = -u_a(s)$.

If we fix $t(D_{i,j}) = 1$, in the example of Fig. 2 the overall cost for the attacker to find the real document is 4 — the cost is instead 5 in the case of Fig. 3.

The above definition assumes the times needed to examine different units are independent. We could in principle remove this assumption and allow such values to be dependent, by setting $u_a(s) = -\sum_{(D_{i,j},\cdot)\in h_s} t(s \mid D_{i,j})$, which reflects that the time spent in examining $D_{i,j}$ is related to the state s including the sequence of units previously examined by the attacker and the observed real document set history.

Given a *strategy profile* (D, y), let $P_{D,y}(s)$ denote the probability that state s is reached. Note that (i) if $h_s \notin H_D$, i.e., s is not reached according to D, then $P_{D,y}(s) = 0$, and (ii) otherwise, $P_{D,y}(s)$ only depends on y and h_s. Therefore, we can simply use $P_y(s)$ to denote the probability of reaching state s under (D, y) with $h_s \in H_D$. Function $P_y(s)$ is recursively defined as:

$$P_y(s_0) = 1. \tag{1a}$$

$$P_y(s) = \sum_{s'\in S\setminus S_t,\, s\in S_{s',a}} P_y(s')y(s', a) \ \forall s \neq s_0. \tag{1b}$$

The first equation above captures the fact that we start from state s_0, so the probability of reaching it is 1. The second equation above says the probability of reaching state s given a behavior strategy y is an expectation, i.e., of the sum of the s of reaching a preceding state s' and the probability of executing an action in state s' that causes state s to result.

The attacker's expected utility then is

$$U_a(D, y) = \sum_{s\in S_t, h_s\in H_D} P_y(s)u_a(s).$$

Similarly, we have

$$U_a(x, y) = \sum_{D\in\mathcal{D}} x(D)U_a(D, y).$$

The defender's expected utility is $U_d(x, y) = -U_a(x, y)$.

E. Defender's Optimal Solution and Equilibrium

In this section, we model the defender's optimal solution via a Stackelberg equilibrium [28], where the defender's optimal strategy is the maximin strategy in zero-sum games. In zero-sum games, the maximin strategy is also a part of the Nash equilibrium [29], [30].

Hence, we compute a Nash equilibrium to obtain the defender's strategy. Generally, $(x^\star, y^\star)$ is a Nash equilibrium [31] if and only if $\forall x, U_d(x^\star, y^\star) \geq U_d(x, y^\star)$ and $\forall y, U_a(x^\star, y^\star) \geq U_a(x^\star, y)$. That is, if the adversary uses strategy $y^\star$, then the defender has no way to gain additional utility by deviating from $x^\star$ — the second equation mirrors this for the attacker.

IV. Computing the Optimal Solution for the Defender

The natural way to solve the defender's problem is via the solution of a nonlinear optimization problem.

A. Nonlinear Program

A Nash equilibrium $(x^\star, y^\star)$ is computed by solving the nonlinear Program (2), where $x^\star$ is obtained by the dual program.

$$\begin{aligned} \max_{y} \quad & U_a \\ \text{s.t.} \quad & U_a \leq U_a(D, y), \quad \forall D \in \mathcal{D} \\ & \sum_{a\in A_s} y(s, a) = 1, \quad \forall s \in S \setminus S_t \\ & y(s, a) \in [0, 1], \quad \forall a \in A_s, \forall s \in S \setminus S_t \end{aligned} \tag{2}$$

Unfortunately, Program (2) is nonlinear and thus hard to solve since each state's $P_y(s)$ (defined in Eq. (1)) involves the product of variables in y.

B. Linear Program Via Probabilistic Flows

In this section, we show how to transform Program (2) into a linear one. Based on the probability conservation property at each state, i.e., the incoming probability is equal to the outgoing probability, we develop a probability flow strategy for the attacker's strategy inspired by the sequence-form strategy in extensive-form games [30], [32].

Suppose $f(s,a)$ specifies the probability that the attacker reaches state $s \in S \setminus S_t$ and takes action $a \in A_s$. Given f and D, the attacker's expected utility is

$$U_a(D,f) = \sum_{s \in S_t : h_s \in H_D} \sum_{s' \in S \setminus S_t : s \in S_{s',a}} f(s',a) u_a(s)$$

where:

$$\sum_{a \in A_{s_0}} f(s_0,a) = 1 \tag{3a}$$

$$\sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} f(s',a) = \sum_{a \in A_s} f(s,a), \forall s \in S \setminus (\{s_0\} \cup S_t) \tag{3b}$$

$$f(s,a) \geq 0, \quad \forall a \in A_s, \forall s \in S \setminus S_t \tag{3c}$$

Moreover, $U_a(x,f) = -U_d(x,f) = \sum_{D \in \mathcal{D}} x(D) U_a(D,f)$.

We now show how to derive the associated probability flow strategy from the behavior strategy and *vice versa*. Given a behavior strategy y, the corresponding flow strategy f is:

$$f(s,a) = P_y(s) y(s,a) \; \forall a \in A_s, \forall s \in S \setminus S_t \tag{4}$$

Given f, the probability of reaching any state is $P_y(s) = \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} f(s',a)$ if $s \neq s_0$ and $P_y(s_0) = 1$, and the corresponding behavior strategy y is:

$$y(s,a) = \begin{cases} \frac{f(s,a)}{P_y(s)} \text{ if } P_y(s) > 0, \\ \frac{1}{|A_s|} \quad \text{otherwise,} \end{cases} \quad \forall a \in A_s, \forall s \in S \setminus S_t. \tag{5}$$

Theorem 1 below ensures that the probability flow strategy yields the same expected utility as the behavior strategy.

Theorem 1: **(a)** Given a behavior strategy y, there is a probability flow strategy f defined by (4) such that $\forall D \in \mathcal{D}$, $U_a(D,y) = U_a(D,f)$. **(b)** Given a behavior strategy f, there is a probability flow strategy y defined by (5) such that $\forall D \in \mathcal{D}$, $U_a(D,y) = U_a(D,f)$.

Proof: **(a)**. Given a behavior strategy y and the corresponding flow strategy of (4), we have $\sum_{a \in A_{s_0}} f(s_0,a) = \sum_{a \in A_{s_0}} P_y(s_0) y(s_0,a) = \sum_{a \in A_{s_0}} y(s_0,a) = 1$, which is according to the second constraint in (2) and (1). By the second constraint in (2) and (1), $\forall s \in S \setminus (\{s_0\} \cup S_t)$, we have $\sum_{a \in A_s} y_{s,a} = \Rightarrow \sum_{a \in A_s} y(s,a) P_y(s) = P_y(s) \Rightarrow \sum_{a \in A_s} P_y(s) y(s,a) = \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} P_y(s') y(s',a) \Rightarrow \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} f(s',a) = \sum_{a \in A_s} f(s,a)$. Therefore, f satisfies Eqs. (3a) and (3b). In addition, we have $U_a(D,y) = \sum_{s \in S_t,\, h_s \in H_D} P_y(s) u_a(s) = \sum_{s \in S_t,\, h_s \in H_D} \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} P_y(s') y(s',a) u_a(s) = \sum_{s \in S_t,\, h_s \in H_D} \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} f(s',a) u_a(s) = U_a(D,f)$. Therefore, $\forall D \in \mathcal{D}$, $U_a(D,y) = U_a(D,f)$.

(b). Given a behavior strategy y and the corresponding flow strategy of Eq. (5), we have $P_y(s_0) = \sum_{a \in A_{s_0}} f(s_0,a) = 1$ and $\forall s' \in S \setminus \{s_0\}$, $P_x(s) = \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} f(s',a)$. In addition, $\forall s \in S \setminus S_t, \forall a \in A_s$,

$$y(s,a) = \begin{cases} \frac{f(s,a)}{P_y(s)} \text{ if } P_y(s) > 0, \\ \frac{1}{|A_s|} \quad \text{otherwise.} \end{cases}$$

Obviously, $\forall a \in A_s$, $\forall s \in S \setminus S_t$, $P_y(s) y(s,a) = f(s,a)$. We have $P_x(s) = \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} f(s',a) = \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} P_y(s) y(s',a)$. Thus, the second constraints in (2) and (1) are satisfied. We further have: $U_a(D,f) = \sum_{s \in S_t,\, h_s \in H_D} \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} f(s',a) u_a(s) = \sum_{s \in S_t,\, h_s \in H_D} \sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} P_y(s') y(s',a) u_a(s) = \sum_{s \in S_t,\, h_s \in H_D} P_y(s) u_a(s) = U_a(D,y)$. Therefore, $\forall D \in \mathcal{D}$, $U_a(D,y) = U_a(D,f)$. ■

Program (6) is the final linear program, equivalent to Program (2).

$$\begin{aligned} &\max_f U_a \\ &\text{s.t.} U_a \leq U_a(D,f), \quad \forall D \in \mathcal{D} \\ &\sum_{a \in A_{s_0}} f(s_0,a) = 1 \\ &\sum_{s' \in S \setminus S_t,\, s \in S_{s',a}} f(s',a) = \sum_{a \in A_s} f(s,a), \forall s \in S \setminus \{s_0\} \setminus S_t \\ &f(s,a) \geq 0, \quad \forall a \in A_s, \forall s \in S \setminus S_t. \end{aligned} \tag{6}$$

This transformation has the nice property that it does not increase the scale of the program when measured by the number of variables and constraints — yet, the nonlinear constraints are reformulated into linear inequalities, making them more amenable to solution via well known LP solvers (it is well-known that solving a linear program is easier than solving a nonlinear one having the same size). Of course, to execute the flow strategy, we still need to transform it into a behavior strategy according to (5).

V. Improving Scalability

Even though one level of scalability was achieved by transforming the nonlinear program for the defender into a linear one, the latter is still huge because the state and action space in our MDP is enormous. In this section, we improve scalability by proposing an incremental strategy generation algorithm.

Our single-oracle GAIT_AO (AO stands for Attacker Oracle) algorithm incrementally expands the attacker's strategy space (by slowly increasing what is considered in the state and the possible actions) until convergence.

Algorithm 1 starts with an initial state without actions (Line 1). At each iteration, GAIT_AO first computes a Nash

Algorithm 1: GAIT_AO.

1 Initialize $S' = \{s_0\}$ and $A' = \emptyset$;
2 **repeat**
3 $(x, f) \leftarrow$ solution of Program (6) under (S', A');
4 $(V, \pi) \leftarrow ABR(s_0, -U_a(x, f), 0)$;
5 **if** $V(s_0) > U_a(x, f)$ **then**
6 **for** *each* s *reached from* s_0 *by* π **do**
7 **if** $s \notin S'$ **then** $S' \leftarrow S' \cup \{s\}$, $A' \leftarrow A' \cup \{\pi(s)\}$;
8 **else**
9 **if** $\pi(s) \notin A'_s$ **then** $A'_s \leftarrow A'_s \cup \{\pi(s)\}$;
10 **until** *convergence:* $V(s_0) \leq U_a(x, f)$;
11 **return** (x, f)

Algorithm 2: *ABR* $(s, b(s), t_s)$ s: current state, b_s: lower bound, t_s: current time.

1 **if** $s \in S_t$ **then**
2 $V(s) \leftarrow u_a(s) \times \sum_{h_s \in H_D} x(D)$;
3 **return** $V(s)$;
4 Sort $a \in A'_s$ descending order of $B(a)$ where $B(a) = \sum_{D \in last\text{-}set(h_s)} -(t_s + B(a, last\text{-}set(h_s), D))x(D)$;
5 $V(s) \leftarrow b(s) - \epsilon$, $\forall a \in A_s$ $V(a) \leftarrow -\infty$;
6 **for** $a \in A_s$ **do**
7 **if** $V(s) + \epsilon \leq B(a)$ **then**
8 Sort $s' \in S_{s,a}$ in ascending order of $B(a, s')$ where $B(a, s') = \sum_{D \in last\text{-}set(h_{s'})} (-(t_s + B(a, last\text{-}set(h_s), D)))x(D)$;
9 //for negative values, we want to find the largest absolute value to cut off $B' \leftarrow \sum_{s' \in S_{s,a}} B(a, s')$;
10 **for** $s' \in S_{s,a}$ *with* $B(a, s') > 0$ **do**
11 $b(s') \leftarrow V(s) + \epsilon - (V(a) + (B' - B(a, s')))$;
12 **if** $b(s') > B(a, s')$ **then break**;
13 **else**
14 $V(s') \leftarrow ABR(s', b(s'), t_s + t(a))$;
15 **if** $b(s') > V(s')$ **then break**;
16 **else** $V(a) \leftarrow V(a) + V(s')$, $B' \leftarrow B' - B(a, s')$;
17 **if** $V(a) < V(s))$ **then**
18 **break**;
19 **if** $V(s) < V(a)$ **then** $V(s) \leftarrow V(a), \pi(s) \leftarrow a$;
20 **return** $V(s)$.

equilibrium for the restricted game (Line 3), and then finds the attacker's best response against the defender's strategy in the Nash equilibrium of the restricted game (Line 4). If the attacker's best response V can improve the attacker's utility, then the corresponding actions and states are added to the restricted game (Lines 5–9). Otherwise, GAIT_AO converges to a Nash equilibrium in the full game (Line 10). It should be observed that Line 3 solves a restricted version of Program (6) where constraints and variables are based on smaller sets S' and A', (i.e, by replacing S with S' and $A = \cup_{s \in S} A_s$ with A' in Program (6)).

A. The Attacker's Best Response Problem

The attacker's best response problem is to compute the best action (i.e., the action that maximizes the attacker's utility) for each state reachable under the defender's mixed strategy x.

We first study the complexity of the associated decision problem: Given a defender's mixed strategy, determine whether there is an attacker strategy such that the cost is at most a given value.

Theorem 2: The attacker's best response problem is NP-hard.

Proof: We show NP-hardness by reduction from SET COVER, that is: Given a set of elements $\mathcal{E}$, a collection $\mathcal{C}$ of subsets of $\mathcal{E}$ and an integer k, determine if there is a cover, i.e., a set $\mathcal{C}_s \subseteq C$ such that $\cup_{c \in \mathcal{C}_s} c = \mathcal{E}$, with $|\mathcal{C}_s| \leq k$. We consider the attacker's best response problem in the $M = 2$ case with $\forall D, D' \in \mathcal{D}$, $first\text{-}real(D, 1) \neq first\text{-}real(D', 1)$ and $|first\text{-}real(D, 2)| = |first\text{-}real(D', 2)| = 1$. Each element $e_i \in \mathcal{E}$ corresponds to a pure strategy $D \in \mathcal{D}$. Each element $c_i \in \mathcal{C}$ corresponds to a document in each $D \in \mathcal{D}$. The relation $e_i \in c_{i'}$ corresponds to the fact that the $real(D_{i',1}) = 1$, i.e., in D, the i'-th document's first unit is real. We assume that at the j-th column, the cost to check a unit is $|\mathcal{D}|^{2(k-1)}$ if the $(k-1)$-th one has been checked. The defender strategy is uniform, i.e., $x(D) = \frac{1}{|\mathcal{D}|}$. Without loss of generality, we assume $|\mathcal{E}| = |\mathcal{D}| > 1$. Now we show that there is a cover of maximum size k if and only if there is an attacker strategy with a cost at most equal to $2|\mathcal{D}|^{2(k-1)} + 1$. The existence of a cover $\mathcal{C}_s \subseteq C$ of maximum size k means that any D's real document set at the first unit will be found at most after k steps. Then, the cost is at most $\frac{\sum_{1 \leq k' \leq k} |\mathcal{D}|^{2(k'-1)} + 1}{|\mathcal{D}|} \leq \frac{(2|\mathcal{D}|^{2(k-1)} + 1)}{|\mathcal{D}|}$ for each $D \in \mathcal{D}$: it suffices to check the units corresponding to elements in $\mathcal{C}_s$ at the first column, and then check the unit in $first\text{-}real(D, 1)$ at the second column. If there is an attacker strategy with a cost not exceeding $2|\mathcal{D}|^{2(k-1)} + 1$, then any D's real document set at the first unit will be found at most after k steps. If there is not a cover $\mathcal{C}_s \subseteq C$ of size k or less, then there is a D such that its real document set at the first unit cannot be found after k steps. Then, there is a cover $\mathcal{C}_s \subseteq C$ of size k or less whose elements correspond to the actions in the attacker's strategy at the first unit. ■

Our best response oracle for the attacker is shown in Algorithm 2.

ABR is a branch-and-bound algorithm that adopts a depth-first approach to searching the state and action space. The algorithm works as follows. Given a state s, if s is a terminal state (Line 1), *ABR* terminates; otherwise, *ABR* estimates bounds and cuts branches if: (1) the upper bound $B(a)$ of action a is less than its lower bound $(V(s) + \epsilon)$ (Line 7); (2) the upper bound $B(a, s')$ of succeeding state s' is less than its lower bound $b(s')$ (Line 12); or (3) the value of state s' is less than its lower bound (Line 15). Then, *ABR* updates $V(a)$ and returns s's value $V(s) = \max_a V(a)$ (Lines 16–20).

Specifically, *ABR* dynamically estimates tight bounds as follows. $B(a, last\text{-}set(h_s), D)$ is the upper bound (formally defined below) by taking action a at state s under D. All actions are sorted according to the upper bound $B(a)$, and an action is evaluated if the upper bound is larger than the lower bound or the maximum value obtained by other actions (Line 7). In addition,

the method initializing $V(s)$ guarantees that the best action at s is computed if the state value is equal to $b(s)$. For each succeeding state s' of action a, the lower bound is computed by assuming that the maximum value is returned by all the remaining succeeding states (Line 11).

Let $j_t(D)$ be lowest index such that $|first\text{-}real(D, j_t(D))| = 1$ (i.e., $|first\text{-}real(D, j_t(D) - 1)| > 1$. We define the minimum time to reach a terminal state as follows:

- The minimum time to reach a terminal state starting from the j-th unit is $B(D, j) = \sum_{j \le j' \le j_t(D)} t(D_{i,j'})$ with $D_i \in first\text{-}real(D, j_t(D))$, and $real(D_{i,j_t(D)}) = 1$ (i.e., the i-th document is real).
- The minimum time to reach a terminal state by taking action a under $first\text{-}real(D, J(a)) \subseteq D' \subseteq D$ with $|D'| > 1$ and $I(a) \in D'$ is $B(a, D', D) = B(D, J(a))$ if $real(a) = 1$; and $B(a, D', D) = B(D, J(a)) + t(a)$ if $real(a) = 0$.

Proposition 1: $B(D, j)$, as defined above, is the minimum time to reach a terminal state starting from the j-th unit, $B(a, D', D)$, as defined above, is the minimum time to reach a terminal state by taking action a under $first\text{-}real(D, J(a)) \subseteq D' \subseteq D$ with $|D'| > 1$ and $I(a) \in D'$.

Proof: The definition of $B(D, j)$ means that at each index j' with $j \le j' \le j_t(D)$, the real version of the j'-th unit is found immediately, which is the minimum time to reach a terminal state. The definition of $B(a, D', D)$ includes the minimum time starting from the $(J(a) + 1)$-th unit and the minimum time to find the real version at the $J(a)$-th unit. $B(D, J(a) + 1)$ is the minimum time to reach a terminal state starting from the $(J(a) + 1)$-th unit. The minimum time to find the real version at the $J(a)$-th unit is the time to check a if $real(a) = 1$ or the time to check a and the time to check a real location at the $J(a)$-th unit if $real(a) = 0$. ■

B. Temporary Terminal States

In the restricted game, GAIT_AO may reach states without actions. For example, in the initial step of GAIT_AO, s_0 does not have actions. However, s_0 is not a terminal state in the full game. To compute a Nash equilibrium in the restricted game, we treat states without actions as *temporary terminal states*. We need to find a way to assign the utility for each player in these temporary terminal states. Note that the defender's utility in the Nash equilibrium in the game with the restricted strategy space of the attacker is an upper bound of the defender's utility in the Nash equilibrium in the full game. Therefore, we can set an upper bound on the defender's utility in a temporary terminal state, i.e., the defender utility under a given attacker strategy that will finally reach terminal states. In this case, if the defender's strategy in the equilibrium of the restricted game will reach this state, GAIT_AO will generate the attacker's best response at this state, and the game is expanded correctly. If we set a lower bound for these temporary terminal states, the defender's utility in the Nash equilibrium in the game with the restricted strategy space of the attacker may not be an upper bound of the defender's utility in the Nash equilibrium in the full game, which may prevent GAIT_AO from converging to a Nash equilibrium in the full game.

Algorithm 3: *EstimatedUtility* (D', j).

1 $\mathcal{E} \leftarrow \{\langle a, \{D \mid real(D_{i,j}) = 1, D \in \mathcal{D}' \mid j_a = j\}\rangle\}$: the locations to be checked and the corresponding strategies in $\mathcal{D}'$ whose unit is true there;
2 Sort a in descending order of $|\mathcal{D}_a|$ with $\langle a, \mathcal{D}_a\rangle \in \mathcal{E}$;
3 $\mathcal{D} \leftarrow \emptyset$, $N \leftarrow 0$;
4 **for** *each* $\langle a, \mathcal{D}_a\rangle \in \mathcal{E}$ **do**
5 $N \leftarrow N + t(a)$;
6 **for** *each* D *in* $\mathcal{D}_a$ **do**
7 $EU_D[j] = N$;
8 $\mathcal{R} \leftarrow \{\langle D', \{D \mid$ *first-real*$(D, j) = D', D \in \mathcal{D}'\}\rangle\}$: the real document sets with the corresponding strategies in $\mathcal{D}'$;
9 **for** $\langle D', \mathcal{D}_{D'}\rangle \in \mathcal{R}$ **do**
10 **if** $|D'| > 1$ **then**
11 *EstimatedUtility*$(\mathcal{D}_{D'}, j + 1)$;

To avoid such issues, we estimate the defender utility at each temporary terminal state s as the cost to reach this state plus the cost to reach a terminal state for each D with $h_s \in H_D$ by following a fixed attacker strategy. The fixed attacker strategy is: at each j_s-th unit with $|last\text{-}set(h_s)| > 1$, and $\mathcal{D}'$ whose strategies generate h_s, we use a set cover method to find the minimum number of actions to find the real document set for each $D \in \mathcal{D}'$. The cost for finding the real document set for $D \in \mathcal{D}'$ is the sum of costs of taking k actions if its real version is found at the k-th action. This method will make sure the real document set for each $D \in \mathcal{D}'$ is found at the j_s-th unit. We will continue this step to the $(j_s + 1)$-th until reaching the terminal state.

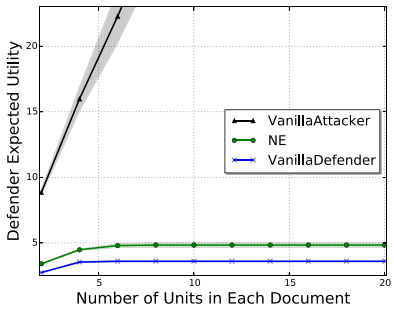
Again, (i) each element $e_i \in \mathcal{E}$ corresponds to a pure strategy $D \in \mathcal{D}$ and (ii) each element $c_i \in \mathcal{C}$ corresponds to a document in each $D \in \mathcal{D}$. The relation $e_i \in c_{i'}$ corresponds to the fact that the $real(D_{i',j_s}) = 1$, i.e., in D, the i'-th document's j_s-th unit is real. We can then use a set cover method to find the minimum number of actions needed to find the real document set for each $D \in \mathcal{D}'$. However, since the set cover problem is NP-hard, we use a greedy algorithm that, at each action, takes the action that corresponds to the largest among all remaining sets.

Based on this idea, Algorithm 3 can compute an estimated utility array $EU_D[j]$ for each $D \in \mathcal{D}$ at the j-th unit before running GAIT_AO. Then, for each temporary terminal state s, the estimated cost to reach a terminal state for D with $h_s \in D$ is $\sum_{j_s \le j \le |EU_D|} EU_D[j]$. Lines 4–7 of Algorithm 3 use the set cover to estimate each element in $\mathcal{D}'$ at the j-th unit as described above. Line 11 transforms to the next unit to estimate the utility if the terminal state is not in the j-th unit.

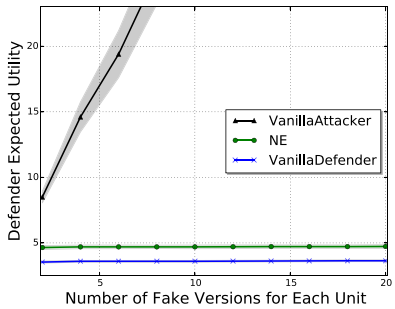
The following result ensures that GAIT_AO successfully converges to the desired Nash equilibrium in the full game.

Theorem 3: GAIT_AO converges to a Nash equilibrium in the full game.

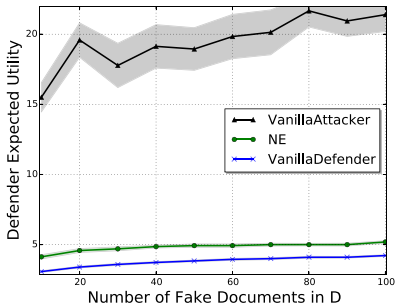
Proof: In GAIT_AO, the attacker state space in the restricted game is reduced, and the estimated defender utility at temporary states is an upper bound. Then, the defender utility value in the Nash equilibrium of the restricted game is an upper bound of the defender utility value in the Nash equilibrium of in the full game, which is a lower bound of the attacker utility value



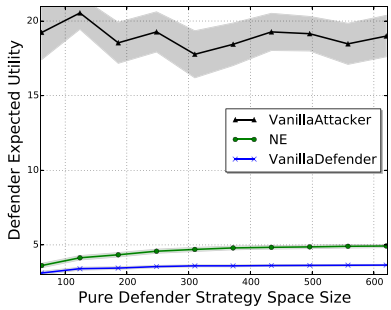
(a) Utility when varying the number of units



(b) Utility when varying the number of fake versions



(c) Utility when varying the number of fake documents



(d) Utility when varying the size of the pure defender strategy

Fig. 4. Defender expected utility.

in the Nash equilibrium in the full game. Therefore, if *ABR* can find an improving strategy, the restricted state is expanded. In the worst case, $S' = S$, and $A' = A$. Therefore, GAIT_AO converges within a finite number of iterations because S and A are finite. When GAIT_AO converges, which means that *ABR* cannot find an improving strategy in the full game, it means that the output (x, f) of GAIT_AO is a Nash equilibrium in the full game. Therefore, GAIT_AO converges to a Nash equilibrium in the full game. ■

VI. Experimental Assessment

These sections report the results of the experiments we carried out to evaluate our proposed approach. Each fake document matrix D was randomly generated. Default values were set to $M = 5$ units, $|F_j| = 5$ possible fakes of each unit, and $|\mathcal{D}| = 31$, i.e., 30 fake documents. Motivated by the algorithm in [4] that generates a set of fake documents for a given real one (they together form D), we take the size of such set as a parameter, i.e., the number of fake documents in $D \in \mathcal{D}$, with the initial value 30. About 80% of units were replaced by fake versions (similarly to the work in [4]). Without loss of generality, we also set $t(a) = 1$ for any action a. The results were averaged over 50 runs with 95% confidence interval. Experiments were run on a 64-bit machine with 16 GB RAM and an 8-core 2.3 GHz processor.

A. Utility

Our first round of experiments was aimed at showing optimality and robustness of our approach: for the Nash equilibrium, if the defender deviates from his equilibrium, he may lose some utility; if the attacker deviates from his equilibrium, the defender may gain some utility. The default size of the pure defender strategy space was 310. We compared the results obtained in the following cases:

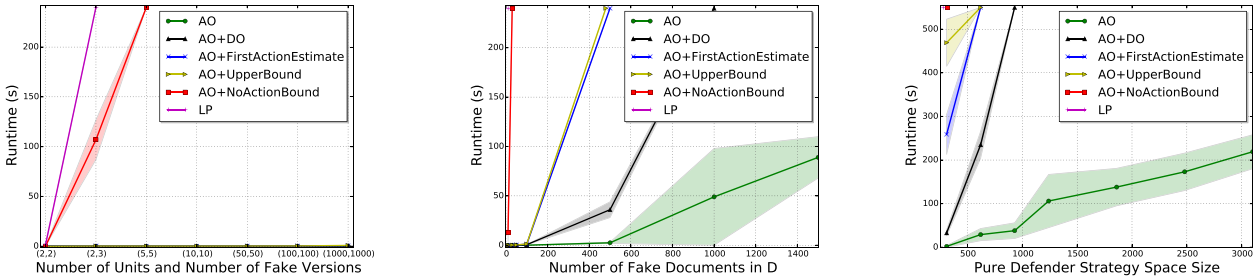
- *NE:* Expected defender utility under the Nash equilibrium strategy profile.
- *VanillaAttacker:* Expected defender utility when the defender plays the Nash equilibrium strategy while the attacker plays a vanilla strategy where he examines the document matrix one row at a time until he discovers the real document. This baseline is used to assess the robustness of our proposed defender strategy.
- *VanillaDefender:* Expected defender utility when the defender plays a uniform strategy in the strategy space while the attacker best responds to it. This baseline is used to assess the optimality of our proposed defender strategy.

Fig. 4 shows the defender expected utility we obtained.

The results show that the defender utility obtained by *NE* are significantly larger than those obtained by *VanillaDefender*. But if the attacker is stupid and plays *VanillaAttacker*, then the defender utility is significantly larger than *NE*. Thus, the defender loses utility if the defender does not play the Nash strategy in the equilibrium, and the defender gains utility if the attacker does not play the Nash strategy in the equilibrium — these results confirm our expectations.

B. Scalability

Our second round of experiments evaluated the scalability of GAIT_AO (*AO* for readability). The default size of the pure



(a) Runtime when varying the number of units and the number of fake versions

(b) Runtime when varying the number of fake documents

(c) Runtime when varying the size of the pure defender strategy

Fig. 5. Runtime (AO means GAIT_AO).

defender strategy space was 31. We employed the following baselines:

- *LP*, using the linear program.
- *AO+NoActionBound*, without the action upper bound in *ABR* according to [24].
- *AO+UpperBound*, just using the maximum step to reach the terminal state to estimate the temporary terminal states. In other words, in this case we only estimate the upper bound of actions, similar to [23].
- *AO+FirstActionEstimate*, using just the first action of each state to estimate the temporary terminal states according to [24].
- *AO+DO*, that is *AO* plus a defender best response oracle starting from the empty pure defender strategy space.

Fig. 5 shows the runtimes we obtained in the above cases.

The results show that *AO* significantly outperforms all baselines — it can solve large problems with thousands of pure defender strategies or elements in the fake document set. In addition, the number of units (M) and the number ($|F_j|$) of versions per unit do not significantly affect scalability. On the other hand, the number of fake documents we seek to generate (i.e., $|D|$) and the size of the pure defender strategy space significantly influence runtime. This is consistent with the results in Fig. 4 as the increase in number of units and versions increases the defender utility initially, then the latter remains stable, and the increase of the number of fake documents in D and pure defender strategy space size significantly increase the defender utility initially, more slowly afterwards. *LP* runs out of memory in most cases. *AO+NoActionBound* takes very long converge because it estimate too many actions in the best response oracle. The inefficiency of *AO+FirstActionEstimate* and *AO+UpperBound* is because their estimation of the defender's upper bound of the utility is too loose. *AO+DO* performs worse than *AO* because the attacker's MDP are generated according to each pure strategy of the defender, which needs more iterations to converge when the defender strategy space is incrementally expanded. We emphasize that GAIT_AO yields very scalable performance compared to all of these alternatives.

VII. Conclusions and Future Work

Statistician George Box famously said "all models are wrong but some are useful." GAIT is no exception to this rule. While many papers have been written on generating fake technical documents, this is the first paper that tries to maximize the expected time that an attacker must spend in order to find the real document from a set of $(n+1)$ similar documents, n of which are fake. Our GAIT framework models how an attacker can target a fake document generation system. We propose a game theoretic strategy for the attacker to explore the documents at minimal cost, and show how the defender can generate a set of fakes in order to maximize the attacker's effort. The proposed GAIT_AO algorithm is shown to perform very well experimentally. A defender who uses GAIT_AO will cause a smart attacker (who plays a Nash equilibrium strategy) to expend maximal possible effort (in expectation) and will beat an attacker who uses any other strategy. Compared to various alternatives, GAIT_AO also performs in a scalable manner, making it suitable for practical use.

Future work may focus on some questions not answered in this paper. (i) What is the best way of splitting a document into units? How should semantics of the document and continuity of narrative flow impact how a document should be split? (ii) In this paper, we have assumed a perfect adversary, one who makes no mistakes when examining a unit and declaring it to be real or fake. How should we generate fake documents if the adversary model assumes some margin of error, e.g., 10% of the adversary's assessments of units are wrong? (iii) While models such as probabilistic logic graphs [5] provide a single framework within which to generate fake multimodal documents containing a mix of text, table, equations, and diagrams, in this paper, we have shown how we can "game out" the adversary's behavior on text data. An important extension would build on ideas such as PLGs and conduct a game-theoretic analysis in the multimodal case. (iv) With the advent of systems such as ChatGPT and GPT-4, there are a few systems to detect content generated by such programs. However, we are not aware of systems to detect fake *technical* documents which can use a lot of complex jargon. Nevertheless, this is an important area for additional research.

References

[1] M. Henriquez, "Winnti APT group stole trillions in intellectual property," *Secur. Mag.*, Jul. 2022. [Online]. Available: https://www.SecurityMagazine.com/articles/97549-winnti-apt-group-stole-trillions-in-intellectual-property

[2] L. Bilge and T. Dumitraş, "Before we knew it: An empirical study of zero-day attacks in the real world," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 833–844.
[3] T. Chakraborty, S. Jajodia, J. Katz, A. Picariello, G. Sperli, and V. Subrahmanian, "A fake online repository generation engine for cyber deception," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 2, pp. 518–533, Mar./Apr. 2021.
[4] A. Abdibayev, D. Chen, H. Chen, D. Poluru, and V. Subrahmanian, "Using word embeddings to deter intellectual property theft through automated generation of fake documents," *ACM Trans. Manage. Inf. Syst.*, vol. 12, no. 2, pp. 1–22, 2021.
[5] Q. Han, C. Molinaro, A. Picariello, G. Sperlì, V. S. Subrahmanian, and Y. Xiong, "Generating fake documents using probabilistic logic graphs," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 4, pp. 2428–2441, Jul./Aug. 2022.
[6] Y. Xiong, G. Ramachandran, R. Ganesan, S. Jajodia, and V. S. Subrahmanian, "Generating realistic fake equations in order to reduce intellectual property theft," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 3, pp. 1434–1445, May/Jun. 2022.
[7] J. Yuill, M. Zappe, D. Denning, and F. Feer, "HoneyFiles: Deceptive files for intrusion detection," in *Proc. Annu. IEEE SMC Inf. Assurance Workshop*, 2004, pp. 116–122.
[8] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.*, 2009, pp. 51–70.
[9] J. B. Michael, G. Fragkos, and M. Auguston, "An experiment in software decoy design," in *Proc. Secur. Privacy Age Uncertainty Int. Conf. Inf. Secur.*, D. Gritzalis, S. D. C. di Vimercati, P. Samarati, and S. K. Katsikas, Eds., 2003, pp. 253–264.
[10] Y. Park and S. J. Stolfo, "Software decoys for insider threat," in *Proc. 7th ACM Symp. Inf. Comput. Commun. Secur.*, 2012, pp. 93–94.
[11] B. Whitham, "Automating the generation of fake documents to detect network intruders," *Int. J. Cyber-Secur. Digit. Forensics*, vol. 2, no. 1, pp. 103–118, 2013.
[12] B. Whitham, "Design requirements for generating deceptive content to protect document repositories," in *Proc. Australian Inf. Warfare Conf.*, 2014, pp. 20–30.
[13] L. Wang, C. Li, Q. Tan, and X. Wang, "Generation and distribution of decoy document system," in *Proc. Int. Conf. Trustworthy Comput. Serv.*, 2013, pp. 123–129.
[14] S. Kang, C. Molinaro, A. Pugliese, and V. S. Subrahmanian, "Randomized generation of adversary-aware fake knowledge graphs to combat intellectual property theft," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 4155–4163.
[15] A. Sinha, F. Fang, B. An, C. Kiekintveld, and M. Tambe, "Stackelberg security games: Looking beyond a decade of success," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 5494–5501.
[16] C. T. Do et al., "Game theory for cyber security and privacy," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–37, 2017.
[17] J. Pawlick, E. Colbert, and Q. Zhu, "A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 1–28, 2019.
[18] A. Iqbal, L. J. Gunn, M. Guo, M. A. Babar, and D. Abbott, "Game theoretical modelling of network/cybersecurity," *IEEE Access*, vol. 7, pp. 154 167–154 179, 2019.
[19] P. Dasgupta and J. Collins, "A survey of game theoretic approaches for adversarial machine learning in cybersecurity tasks," *AI Mag.*, vol. 40, no. 2, pp. 31–43, 2019.
[20] M. Jain, J. Pita, M. Tambe, F. Ordónez, P. Paruchuri, and S. Kraus, "Bayesian stackelberg games and their application for security at Los Angeles international airport," *ACM SIGecom Exchanges*, vol. 7, no. 2, pp. 1–3, 2008.
[21] M. Jain, D. Korzhyk, O. Vaněk, V. Conitzer, M. Pěchouček, and M. Tambe, "A double Oracle algorithm for zero-sum security games on graphs," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2011, pp. 327–334.
[22] Y. Zhang, B. An, L. Tran-Thanh, Z. Wang, J. Gan, and N. R. Jennings, "Optimal escape interdiction on transportation networks," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 3936–3944.
[23] Y. Zhang, Q. Guo, B. An, L. Tran-Thanh, and N. R. Jennings, "Optimal interdiction of urban criminals with the aid of real-time information," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 1262–1269.
[24] B. Bosansky, C. Kiekintveld, V. Lisy, and M. Pechoucek, "An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information," *J. Artif. Intell. Res.*, vol. 51, pp. 829–866, 2014.
[25] T. B. Brown et al., "Language models are few-shot learners," 2020, *arXiv: 2005.14165*.
[26] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics Hum. Lang. Technol.*, 2019, pp. 4171–4186.
[27] F. Mathey, F. Mercier, M. Spagnol, F. Robin, and V. Mouries, "6, 6'-bis-(1-phosphanorbornadiene) diphosphines, their preparation and their uses," U.S. Patent 6,521,795, Feb. 18, 2003.
[28] V. Conitzer and T. Sandholm, "Computing the optimal strategy to commit to," in *Proc. ACM Conf. Electron. Commerce*, 2006, pp. 82–90.
[29] J. Von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton, NJ, USA: Princeton Univ. Press, 1953.
[30] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[31] J. Nash, "Non-cooperative games," *Ann. Math.*, vol. 54, pp. 286–295, 1951.
[32] B. von Stengel, "Efficient computation of behavior strategies," *Games Econ. Behav.*, vol. 14, no. 2, pp. 220–246, 1996.

Youzhi Zhang received the bachelor's and master's degrees from the Sun Yat-sen University, China and the PhD degree in computer science from the Nanyang Technological University, Singapore. He is an assistant professor with the Centre for Artificial Intelligence and Robotics, Hong Kong Institute of Science & Innovation, Chinese Academy of Sciences. From 2021 to 2022, he was a postdoctoral researcher with the Northwestern University, USA. From 2020 to 2021, he was a postdoctoral researcher with the Dartmouth College, USA. His research interests include artificial intelligence, multi-agent systems, computational game theory, multi-agent reinforcement learning, security, optimization, and so on.

Dongkai Chen received the master's degree in computer science from Dartmouth College, USA. He is a machine learning software engineer with Google LLC, USA. From 2019 to 2021, he was a research assistant working with Professor V.S. Subrahmanian. His research interests include generative artifical intelligence, natural language understanding, data mining, and so on.

Sushil Jajodia (Life Fellow, IEEE) is Distinguished University Professor, BDM International professor, and the founding director of Center for Secure Information Systems with the College of Engineering and Computing, the George Mason University, Fairfax, Virginia. He is also the director of the NSF IUCRC Center for Cybersecurity Analytics and Automation (CCAA). He has made research contributions to diverse aspects of security and privacy, including access control, multilevel secure databases, vulnerability analysis, moving target defense, cloud security, and steganography, as well as replicated and temporal databases and algebraic topology. He has authored or coauthored seven books, edited 53 books and conference proceedings, and published more than 500 technical papers in the refereed journals and conference proceedings. He is also a holder of 28 patents, 17 of which have been licensed by successful startups. He is a fellow of ACM and IFIP; and recipient of numerous awards including the IEEE Computer Society W. Wallace McDowell Award. According to the Google Scholar, he has more than 50,000 citations and his h-index is 115. He has supervised 27 doctoral dissertations; thirteen of these graduates hold academic positions while rest are in successful industrial positions.

Andrea Pugliese (Member, IEEE) received the PhD degree in computer and systems engineering from the University of Calabria, Italy in 2005. Currently, he is an associate professor with the DIMES Department, University of Calabria. His main research interests include computer security (specifically, intellectual property protection, fraud detection in reviewing systems, multi-agent systems, activity detection, and adversarial defense of enterprise systems) and graph data management. He is a member of ACM.

V. S. Subrahmanian is the Walter P. Murphy professor of Computer Science and Buffet faculty fellow with the Buffet Institute of Global Affairs, Northwestern University. He was previously the Dartmouth College distinguished professor in Cybersecurity, Technology, and Society and director of the Institute for Security, Technology, and Society with Dartmouth. Earlier, served as a professor of Computer Science with the University of Maryland from 1989-2017 where he also served for 6+ years as director of the University of Maryland's Institute for Advanced Computer Studies. He is one of the world's foremost experts with the intersection of AI and security issues. He pioneered the development of machine learning and AI-based techniques to analyze counter-terrorism, cybersecurity, text, geospatial, and social network based data in order to generate forecasts of various types of outcomes. He has written 8 books, edited 10, and published more than 300 refereed articles. He is an elected fellow of the American Association for the Advancement of Science and the association for the Advancement of Artificial Intelligence and received numerous other honors and awards. His work has been featured in numerous outlets such as the Baltimore Sun, the Economist, the Wall Street Journal, Science, Nature, the Washington Post, American Public Media and more. He serves on the editorial boards of numerous journals including *Science*, and currently serves on the Board of Directors of SentiMetrix, Inc. and on the Research Advisory Board of Tata Consultancy Services. He previously served on the Board of Directors of the Development Gateway Foundation (set up by the World Bank), DARPA's Executive Advisory Council on Advanced Logistics and as an ad-hoc member of the US Air Force Science Advisory Board.

Yanhai Xiong received the PhD degree in computer science from Nanyang Technological University. She is an assistant professor of Data Science with William & Mary, after which she accepted a postdoctoral position in 2018 with Dartmouth College and assistant professor position in 2022 with University of Louisville before coming to William & Mary. Her research focuses on Artificial Intelligence for cyber-physical systems, including decision making and cybersecurity.