Robust and Resilient Federated Learning for Securing Future Networks

Yushan Siriwardhana*, Pawani Porambage*, Madhusanka Liyanage†*, Mika Ylianttila*,
*Centre for Wireless Communications, University of Oulu, Finland. email: {firstname.lastname}@oulu.fi
†School of Computer Science, University College Dublin, Ireland. email: madhusanka@ucd.ie

***Abstract*—Machine Learning (ML) and Artificial Intelligence (AI) techniques are widely adopted in the telecommunication industry, especially to automate beyond 5G networks. Federated Learning (FL) recently emerged as a distributed ML approach that enables localized model training to keep data decentralized to ensure data privacy. In this paper, we identify the applicability of FL for securing future networks and its limitations due to the vulnerability to poisoning attacks. First, we investigate the shortcomings of state-of-the-art security algorithms for FL and perform an attack to circumvent FoolsGold algorithm, which is known as one of the most promising defense techniques currently available. The attack is launched with the addition of intelligent noise at the poisonous model updates. Then we propose a more sophisticated defense strategy, a threshold-based clustering mechanism to complement FoolsGold. Moreover, we provide a comprehensive analysis of the impact of the attack scenario and the performance of the defense mechanism.**

***Index Terms*—Federated Learning, Poisoning Attacks, Defense Mechanism, Label Flipping**

I. Introduction

Artificial intelligence (AI) and Machine Learning (ML) play vital roles in future mobile networks concerning many aspects such as intelligent wireless communications, big data analytics [1], optimization of network architectures, protocols and operations [2], intelligent attack detection, mitigation, and prevention. The typical setting in ML considers a centralized learning phase using a centralized dataset collected from multiple distributed agents. However, the centralized nature in the conventional ML has specific issues in terms of lack of privacy and inefficient utilization of network bandwidth. Since the real-world data is usually decentralized across many entities, it is challenging to collect data continuously to process at one central server. As the networks bring intelligence towards the edge of the network, running the ML algorithms at the edge is also required. This requires data collection from multiple sources and increases the need for having distributed ML algorithms over centralized ones more than ever.

The complexity of 5G and 6G wireless networks is immense due to billions of connected devices, interconnection of a vast number of heterogeneous networks including hyper-connected clouds, and the realization of novel applications [3]. The difficulty of operation and management of complex next generation networks motivates the automation of networks. One such initiative that leverages future networks is the Zero-touch Network and Service Management (ZSM) proposed by European Telecommunications Standards Institute (ETSI) [4]. Deployment of AI/ML techniques is mandatory to enable automation and the distributed nature of future networks calls for a distributed implantation of AI/ML techniques. Federated Learning (FL) is a decentralized ML technique that can be easily adopted in future networks.

The end devices (i.e., workers) in FL can be vastly distributed and they participate in a learning process with a centralized entity (i.e., parameter server) by training a model shared by the centralized entity. The workers train the shared model locally using the local data and transmit the trained model back to the parameter server. The parameter server combines the received models and shares the aggregated model back to the workers in an iterative manner. The local data regime ensures data privacy and communication efficiency.

FL environment is inherently insecure due to the poisonous model parameters supplied to the centralized server [5]. Since the parameter server cannot guarantee that the workers provide accurate local models, defense mechanisms are a must for robust FL before applying FL techniques for 5G and future 6G networks. The server must have robust techniques to distinguish poisonous and honest users and learn only from honest users. This is challenging as the parameter server does not possess any validation data in a practical scenario.

Our work improves the robustness of FL in the presence of noisy adversaries. We explore the limitations of the state-of-the-art robust algorithm FoolsGold [6], perform an intelligent noise attack using coordinated adversaries to circumvent its defense. We demonstrate the relation between the noise level and the attack success. We also propose a modified, more sophisticated threshold-based clustering defense algorithm to complement FoolsGold. We show that our algorithm performs better in the presence of noisy adversaries and provides a similar performance as FoolsGold when there are no adversaries. The remainder of the paper is organized as follows: Section II presents the related work on robust FL for 5G and beyond networks. Section III describes the threat model and the attack methodology. Section IV evaluates the success of the attack. Section V describes the novel defense mechanism we propose. Finally, Section VI concludes the paper with the future research directions.

II. Related Work

FL provides vital solutions to achieve ubiquitous AI in 6G [7]. The use of FL for the next-generation networked industrial systems [8], ultra-reliable low-latency vehicular communications [9] presents the significance of FL for future networks. AI act as an enabling technology that improves ZSM performance [10]. Despite being an enabler for ZSM,

AI introduces new limitations and risks that need to be addressed to make ZSM a reality [11].

FL systems are vulnerable to model poisoning attacks [5] because the end users possess the data. Byzantine-Robust FL proposes solutions for the model poisoning attacks on FL systems. Krum [12], Bulyan [13], trimmed mean [14] and median [14] algorithms distinguish poisonous users and eliminate them from the learning process. These algorithms operate under the principle of "adversaries deviate the learning process from a common goal", and discard highly dissimilar model updates. However, Krum, Bulyan, and trimmed mean require an estimation of the number of adversaries in the system, thus, limiting their practical applications. The median algorithm is easily vulnerable to increasing attacker count. Moreover, all these algorithms are vulnerable to targeted attacks [5]. FoolsGold algorithm [6] achieves substantially improved performance over the existing algorithms without relying on the expected number of adversaries or a validation dataset at the server. It considers that the adversaries have a common goal to deviate from the learning process. Hence, they collectively act similarly. The algorithm uses update similarity as a key parameter to distinguish poisonous updates and performs better with many adversaries. However, it has poor performance with a small number of adversaries in the system, vulnerable to intelligent and dynamic noise attacks.

III. Threat Model and Attack

A. System model

The system model consists of a central server and a decentralized set of nodes as in a general FL system. The adversary is an entity with the intention of performing a targeted poisoning attack on the FL system. The adversary controls C poisonous nodes (attackers) that perform the label flipping attack. Honest nodes do not perform label flipping attacks and provide accurate local model updates to the central server. The total number of nodes in the system is n. The adversary does not know the count of honest nodes in the system but fully controls the C poisonous nodes and performs coordinated attacks. The adversary knows that the central server is equipped with FoolsGold algorithm, which introduces robustness against poisoning attacks on FL systems. Figure 1 depicts various components of the system model.

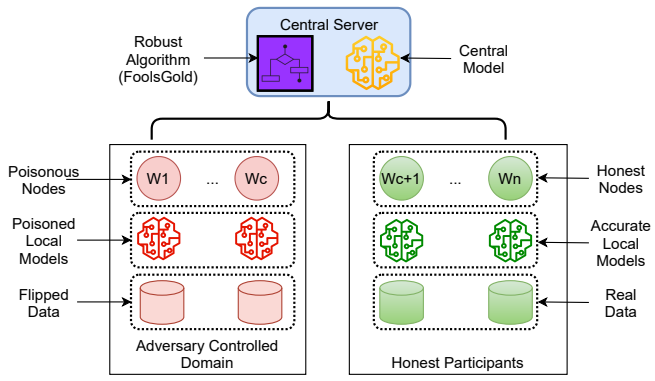

Fig. 1: System model of poisoning attacks for FL

We exploit the weaknesses of the existing FoolsGold design and perform a coordinated, targeted model poisoning attack to circumvent its defense. We use the same ML model used in the original paper [6], a single layer fully-connected softmax for classification. We use MNIST digit classifier dataset [15] for two reasons. First, the original paper uses MNIST dataset to illustrate the performance of the FoolsGold design. Second, MNIST dataset is widely used for performance evaluation of ML algorithms in general. As we are demonstrating an attack against a robust FL algorithm, using MNIST provides a good benchmark for future research. We use the following assumptions about the FL system to perform the attack.

1) The adversaries know that the FoolsGold algorithm is implemented at the parameter server to distinguish poisonous updates.
2) The training data contains a set of features that are not vital for the accuracy of the model training.
3) The adversary controls a sufficient number of users/nodes of the system to launch a coordinated attack, known as the poisonous nodes.

B. Intelligent noise attack

The first step of the attack is flipping the labels of a chosen class. We call the original class the source class and the flipped class as the target class. We denote this attack as (source→target) attack. For example, an attack that flips the labels of class 1 to class 7 is called (1→7) attack. We do not flip the labels of other classes to make sure the server has less probability of detecting the attack, as other classes behave normally. As FoolsGold algorithm primarily depends on the Cosine Similarity (CS) of the received gradient updates as depicted in equation 1, we intelligently introduce random noise to the gradient updates at poisonous nodes. The poisonous nodes train the local models with the label flipped data to obtain a poisonous gradient update. Then they add the noise to the poisonous gradient update before sending them to the central server. A noise vector specifically replaces a part of a given poisonous gradient update to introduce extra dissimilarity among other noisy poisonous gradient updates. We explain the algorithm for the intelligent noise addition later in Section III-B. Only the gradient updates corresponding to the less important features of the model are replaced with noise, while the vital features required for model training are kept as they are. This ensures that the noise does not reduce the effectiveness of the attack. The Cosine Similarity (CS) between two gradient updates $\mathbf{A}$ and $\mathbf{B}$ can be expressed as follows,

$$Cosine\ Similarity\ (\mathbf{A},\mathbf{B}) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \quad (1)$$

Addition of the intelligent noise: As the adversary has access to the data of all the poisonous nodes, he possesses prior knowledge on which gradient updates are vital for attack and which gradient updates can be populated with noise. Therefore, after the local model training is completed, the adversary replaces x amount of gradient values in the local

model updates. These gradient values are less significant to the model training. As an example, the ones corresponding to the black pixels of the edges of MNIST dataset images.

First, we separate the poisonous nodes into groups of four. The reason to create groups of four poisonous nodes is to have a minimal similarity among group members while having substantial amount for coordination within the group. We generate a noise vector N_1 comprising x features, using standard normal distribution. We generate the second noise vector N_2 orthogonal to N_1. We also create the negatives of those vectors, $-N_1$ and $-N_2$. The cosine similarity among these four vectors are $CS(N_1, N_2) = 0$, $CS(-N_1, -N_2) = 0$, $CS(N_1, -N_1) = -1$ and $CS(N_2, -N_2) = -1$. We then multiply each of these four vectors by a scalar called Noise Intensity (I). The scalar multiplication does not affect the cosine similarity. Noise intensity provides dominance for the noise values over the other gradient values in a trained model once inserted. Noise intensity (I_i) for a given poisonous node i is the multiplication of two parameters as shown in equation 2, the mean of the absolute values of gradient update M_i of a poisonous node and an integer value called Noise Scale (N). By varying N, we can obtain different I_i for a given poisonous node i at different instances. Since I_i depends on M_i, a higher N makes sure the dominance of noise over the other gradient values generated after local training.

$$I_i = N.M_i \tag{2}$$

We then replace specifically selected gradient values of poisonous node 1 in a given group by the values inside $I.N_1$. We repeat the same process for the other three poisonous nodes using the noisy vectors $I.N_2$, $-I.N_1$ and $-I.N_2$ respectively. The noise values have a higher effect on the cosine similarity due to I, even though the full gradient vectors are not orthogonal or negative to each other. The domination of noise makes sure that the cosine similarity is kept at a lower value. This way, the four users of a given group have minimum cosine similarity. We repeat the same procedure for all the remaining groups. Algorithm 1 depicts the process for intelligent noise addition. A different strategy is to form the groups of two or more than four poisonous nodes, which we do not discuss in this paper and consider as future work.

As N increases, the FoolsGold algorithm decides that the poisonous nodes exert a highly dissimilar behavior. This causes the honest users to have a higher cosine similarity than the poisonous nodes, reducing their impact on the next global model update. A significantly higher value for noise intensity makes the system learn only from the poisonous nodes as they have an extreme dissimilarity. It makes the system learn faster, only using the poisonous nodes, and causes the Test Accuracy (TA) of attacked class to decline faster.

IV. Evaluation of the Attack

We input MNIST digit classification dataset to evaluate the effectiveness of the attack [15]. We consider IID data distribution at each node and analyze different data distributions in the future work. Since the focus is on targeted poisoning attacks, we perform a (1→7) attack as in the original paper [6]. For the attack analysis of this paper, we keep the total number of users (n) at 40 and consider the number of poisonous nodes (C) as 4, 8, 12, 16 and 20. We replace 10% of the gradient values of the gradient update with noise, corresponding to the less important features for the model training. Figure 2 depicts the test accuracy of FoolsGold algorithm in the presence of a (1→7) label flipping attack without noise (with $C = 12$), which we use for the performance comparison of attack with intelligent noise. We obtained similar results for other C values also, even though they are not shown. The algorithm is robust against the label flipping attack as the test accuracy of the source class (class 1) is not affected by the attack. We use two measurements to determine the effects of the attack.

Algorithm 1: Attack FoolsGold with Intelligent Noise

Data: Local SGD update Δ_i of each attacker i, Attacker count C, Noise intensity I, Indexes of gradient values to be replaced
Result: Noisy gradient vectors of attackers
Attacker groups = $C \div 4$;
for *Each attacker group* **do**
  Create noise vector N_1;
  Generate N_2 such that $N_1 \perp N_2$;
  Generate $-N_1$ and $-N_2$;
  Create $I.N_1$, $I.N_2$, $-I.N_1$, $-I.N_2$;
  Replace the selected values of Δ_j of attacker j with noise values from $I.N_1$, $I.N_2$, $-I.N_1$, $-I.N_2$;
end

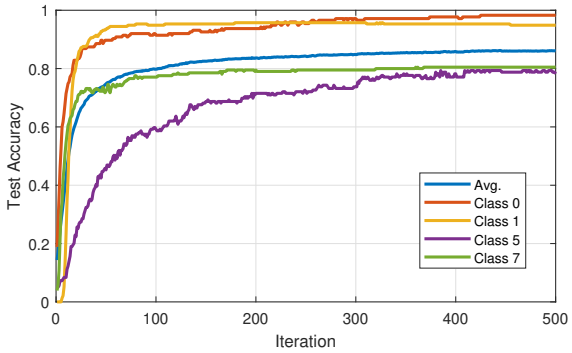

Fig. 2: FoolsGold test accuracy during a label flipping attack with $C = 12$

1) The test accuracy of the source class at the convergence. This determines whether the attack is successful or not.
2) The number of iterations taken for test accuracy of source class to become zero. This determines the magnitude of the attack.

A. Evaluating the success of the attack

A successful attack causes the test accuracy of the source class to decline and to ultimately reach zero. Test accuracy converges to a value close to 1 in a failed attack. We measure the effectiveness of our noise attack using the attack success probability (P_{att}), which is the ratio between the successful attacks count to total attack attempts. P_{att} for varying N from 0 to 150 is depicted in Figure 3 for different C values we consider. Each reported data point is the average of 20 experiments. For a fixed C, at lower N values, the attack fails while the attack becomes successful when N is increasing. For further increased N values, P_{att} reaches 1 and remains as it is. Figure 3 also shows that, as C increases (denoted by different lines) P_{att} reaches 1 at a significantly lower N. At higher C values, a small increase in N causes larger increases in P_{att} and it quickly reaches 1. Figure 3 also shows that the gap between different C lines keeps increasing while C in increasing. It means, at higher C values, increasing the attacker count by 4 has a greater effect on P_{att} than at lower higher C values. For example, the increase the attackers from 16 to 20 has a higher effect than increasing the attackers from 4 to 8, even though in both cases the increased attacker count is 4.

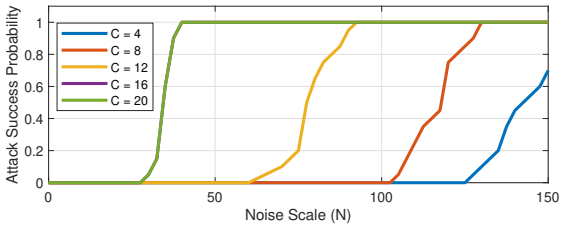

Fig. 3: Attack success probability against noise scale

Test accuracy of a successful (1→7) attack with $C = 12$ for a selected suitable N (N=100) is depicted in Figure 4. As the system learns, the test accuracy of the source class (class 1) reaches 0. There is a noticeable reduction in target class (class 7) test accuracy also. The test accuracy of other classes remain similar to Figure 2, which means those classes are not affected. Average accuracy also shows a decrease compared with Figure 2 mainly due to zero class 1 test accuracy. Figure 5 shows the test accuracy of two attacks with $C = 12$, one with N = 100 and the other with N = 10. The increase of N leads to the attack's success.

B. Evaluating the magnitude of the attack

We analyze the magnitude of the attack when $C = 12$ for the N values that result in $P_{att} >$ 0.5. We calculate the average number of iterations taken for the test accuracy to reach 0. Attack success with fewer iterations means that the attack has a higher impact. Table I shows that when N is high, the system starts to predict the source class incorrectly with fewer iterations.

V. Defending against Noisy Adversaries

We propose implementing a defense technique at the central server to prevent intelligent noise attacks from co-

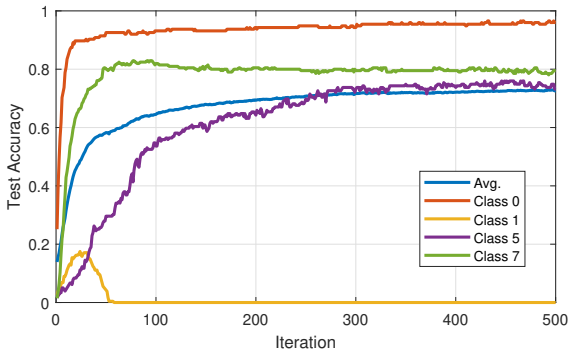

Fig. 4: TA of a successful attack with $C = 12$ and $N = 100$

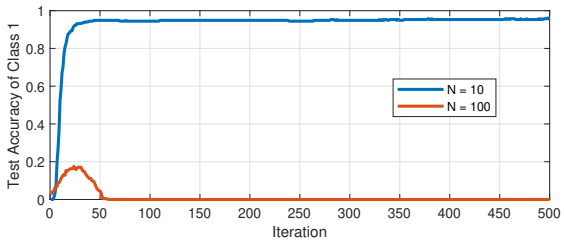

Fig. 5: TA of class 1 for different noise scales with $C = 12$

ordinated attackers. To the best of our knowledge, the defense strategy against noise attacks is to use a weighing mechanism at the central server, considering the importance of model parameters. The weighing mechanism assigns a higher weight for indicative features and a lower weight for non-indicative features at the model aggregation. The weighing mechanism reduces the noise added to the non-indicative features. However, the adversary may also add noise to the indicative features, limiting the performance of the weighing mechanism. It is worth noting that the adversary compromises attack effectiveness to a certain extent by adding noise to the indicative features. Also, to learn the indicative features, the central server must have access to a certain amount of data. Due to these limitations, we propose implementing a novel threshold-based mechanism to identify the noisy updates.

A. Defense mechanism

The defense mechanism operates on the principle that the magnitude of the noise values is significantly higher than the other gradient values due to N. As shown in Figure 3, higher N results in a better P_{att}. Then, the mean of magnitude of the gradient updates from poisonous nodes is significantly higher than the mean of the honest participant's gradient updates. As N increases, this difference should become more dominant such that the server can cluster the gradient updates into two groups, noisy updates, and honest updates.

The intuition of the defense mechanism is, for higher values of N, it should use the clustering mechanism to differentiate the noisy and honest updates. In contrast, for lower values of N, it should use the FoolsGold algorithm. The defense mechanism operates as follows. The central server separates the gradient updates from all the users into

TABLE I: Average no. of iterations needed against N

N	80	85	90	95	100	105	110
Iterations	168.3	140.7	114.8	83.1	53.0	43.6	35.8

two clusters based on the mean of magnitudes of all gradient values of a gradient update. Then the server calculates the ratio (R) between the mean values of gradient updates of the two clusters. We always take the ratio between the higher mean value over the lower mean value, therefore $R \geq 1$. Suppose R is sufficiently high. In that case, there are two distinct clusters, and the higher mean value corresponds to the adversary group. Then they can be eliminated in the next global model update. If R is low, a clear separation is not visible, and FoolsGold will be used to defend. The defense mechanism uses a threshold (K) to decide whether to use the cluster-based separation or use FoolsGold at the central server.

B. Analysis of defense results

We implement this defense mechanism and conduct simulations for the range of N that has the $P_{att} > 0.5$. We keep C fixed at 12 for this analysis and consider other cases in future work. Table II shows the defense success probability (P_{def}) for different K under different N values. P_{def} is the ratio between the number of successful defenses to total defense attempts. Once the defense mechanism is implemented, if the test accuracy of the source class converges towards 1, it is a successful defense. If the test accuracy declines and reaches zero, then it is a failed defense. Each reported data point is the average of 20 experiments.

TABLE II: Defense probability against N and K

N	K								
	1-1.5	**1.6**	**1.7**	**1.8**	**1.9**	**2.0**	**2.1**	**2.2**	**2.3**
120	1.0	1.0	0.95	0.85	0.6	0.1	0.05	0.0	0.0
115	1.0	1.0	0.9	0.7	0.25	0.1	0.1	0.0	0.0
110	1.0	1.0	0.7	0.55	0.15	0.15	0.0	0.0	0.0
105	1.0	1.0	0.65	0.3	0.05	0.05	0.0	0.0	0.0
100	1.0	1.0	0.6	0.4	0.15	0.05	0.05	0.0	0.0
95	1.0	1.0	0.75	0.4	0.25	0.25	0.15	0.15	0.0
90	1.0	0.9	0.9	0.55	0.2	0.2	0.2	0.2	0.15
85	1.0	1.0	0.95	0.65	0.6	0.65	0.6	0.5	0.2
80	1.0	1.0	0.95	0.8	0.8	0.85	0.75	0.65	0.2

1.0 | 0.5 - 1.0 | 0.0 - 0.5 | 0.0

First, consider the row corresponding to N=120. For $1 \leq K \leq 1.5$ the defense mechanism always performs with P_{def}=1. As K increases, P_{def} declines, which means the performance of the defense mechanism is decreasing. This is because, for a given N, R is a constant, and if the defense mechanism sets a higher K, then the algorithm chooses FoolsGold as the algorithm at the central server. However, it is shown in Figure 3 that at this range of N values, FoolsGold fails. Therefore, if N is high, the defense mechanism must make sure to differentiate the poisonous nodes using the clustering method by setting $K \leq R$. As R is proportional to N, when N is further reduced (for example N=105), the threshold also should be further reduced to achieve better P_{def} values.

As N reduces further, the behavior changes. Because at lower N, FoolsGold starts to demonstrate robustness as shown in Figure 3. Even though R is a lower value at this range of N, it is not mandatory to set $K \leq R$. Even if K is set to a higher value than R, FoolsGold algorithm's mechanism distinguishes the noisy gradient updates. For example, at N = 10, which is not shown in the Table II, it does not matter if K is set to a very high value because FoolsGold algorithm performs better at this range of N.

A notable factor here at this range of N is, setting a far low K would force the system to use clustering. As there are no two distinct groups for lower N, the system may penalize certain honest users and consider certain adversaries to calculate the model update of the next iteration. Therefore K should be a dynamic value for each iteration. Setting a dynamic value for K will be considered in future work.

C. Performance comparison with FoolsGold

Based on K, the defense mechanism classifies users as either poisonous or honest. We test the effectiveness of our defense mechanism under three scenarios for the range of K that produces P_{def}=1, which is $1 \leq K \leq 1.5$. First, for a system without poisonous nodes, second, for a system where poisonous nodes perform a label flipping attack, and third, a system with intelligent noise attack. We compare our defense mechanism with FoolsGold [6], considering the test accuracy and the number of iterations needed to converge. The analysis aims to check whether our algorithm achieves a similar performance as FoolsGold.

1) A system with no attackers: Results of the comparison are outlined in Table III. Each reported data point is an average of 20 experiments. Our defense mechanisms achieve similar test accuracy in a similar number of iterations.

TABLE III: Evaluation for a system with no poisonous nodes

K	**Test Accuracy of Class 1**		**Avg. Iterations Needed**	
	FoolsGold	**Our Algorithm**	**FoolsGold**	**Our Algorithm**
1.1	0.947	0.951	55.4	68.3
1.2		0.943		59.2
1.3		0.949		58.9
1.4		0.947		57.8
1.5		0.945		54.6
1.6		0.943		59.1

2) For a label flipping attack: As outlined in Table IV, our defense mechanism performs similarly to FoolsGold in many cases. But for lower K, our defense mechanism takes more iterations. This may be due to switching between the clustering-based defense and FoiolsGold due to lower R.

TABLE IV: Evaluation for a label flipping attack

K	**Test Accuracy of Class 1**		**Avg. Iterations Needed**	
	FoolsGold	**Our Algorithm**	**FoolsGold**	**Our Algorithm**
1.1	0.947	0.941	67.65	107.4
1.2		0.941		93.8
1.3		0.927		78.8
1.4		0.939		71.9
1.5		0.945		63.0
1.6		0.941		59.1

3) For a label flipping attack with noise: For $80 \leq N \leq 120$, our algorithm achieves similar test accuracy compared with scenario 1 and 2. FoolsGold either fails or achieves poor results in terms of both the test accuracy (Table V) and the number of iterations needed (Table VI) to converge. P_{att} is also depicted alongside the results.

TABLE V: Test accuracy during a label flipping + noise attack

N	Our Algorithm (K)					FoolsGold	
	1.1	**1.2**	**1.3**	**1.4**	**1.5**	**Acc.**	P_{att}
120	0.949	0.944	0.928	0.949	0.949	0.0	100%
115	0.948	0.947	0.948	0.940	0.946	0.0	100%
110	0.947	0.947	0.941	0.941	0.947	0.0	100%
105	0.942	0.949	0.939	0.942	0.945	0.0	100%
100	0.946	0.946	0.945	0.940	0.950	0.0	100%
95	0.947	0.947	0.949	0.944	0.947	0.0	100%
90	0.941	0.944	0.943	0.945	0.948	0.894	95%
85	0.940	0.944	0.957	0.948	0.952	0.903	80%
80	0.946	0.948	0.944	0.942	0.946	0.899	65%

TABLE VI: Average number of iterations needed during a label flipping + noise attack

N	Our Algorithm (K)					FoolsGold	
	1.1	**1.2**	**1.3**	**1.4**	**1.5**	**Iterations**	P_{att}
120	94.8	78.5	84.5	81.8	74.9	–	100%
115	77.7	83.2	69.1	71.6	71.9	–	100%
110	78.2	73.9	72.3	74.7	79.5	–	100%
105	67.4	71.4	61.8	70.2	72.3	–	100%
100	60.0	72.6	61	65.2	65.6	–	100%
95	83.5	71.2	68.1	65.7	79.4	–	100%
90	70.9	71.1	71	71.7	81.2	95.5	95%
85	70.8	71.4	68.1	76.3	77.2	110.8	80%
80	86.45	83.3	71.6	82.1	72	119.1	65%

VI. Conclusions

The applicability of AI and ML techniques is immense in 5G and 6G networks, especially with future network automation. Federated Learning (FL), a distributed machine learning technique, shows great potential as the networks become more distributed in the future. Hence, vulnerabilities of FL should be explored, and techniques should be introduced to enhance the robustness of FL against attacks, before its deployment in real networks. In this paper, we perform a poisoning attack with intelligent noise to circumvent the defense of the FoolsGold algorithm designed to enhance the robustness of FL. We evaluate the success probability and the magnitude of our attack and demonstrate the effectiveness of the attack. We also propose a sophisticated defense mechanism that uses a threshold-based clustering mechanism to compliment the FoolsGold algorithm. We also evaluate the performance of the novel defense mechanism, compare it with FoolsGold, and show that our defense mechanism performs better under intelligent noise attacks.

As future work, we will consider different grouping strategies for attackers to create more efficient attacks. Moreover, we will analyze the relationship between the number of attackers and the noise scale to optimize the attack further. In addition to that, we will also consider more dynamic noise insertion strategies. On the defense side, we will investigate the possibility of dynamic shareholding to add more sophisticated defense in the presence of more optimized poisoning attacks.

Acknowledgement

This work is supported by Academy of Finland in 6Genesis Flagship (grant no. 318927) project. The research leading to these results partly received funding from European Union's Horizon 2020 research and innovation programme under grant agreement no 871808 (5G PPP project INSPIRE-5Gplus) and 101021808 (H2020 SPATIAL project). The paper reflects only the authors' views. The Commission is not responsible for any use that may be made of the information it contains.

References

[1] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "AI and 6G Security: Opportunities and Challenges," *2021 IEEE Joint European Conference on Networks and Communications (EuCNC) 6G Summit, 2021*, pp. 1–6, 2019.

[2] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, "The Roadmap to 6G: AI Empowered Wireless Networks," *IEEE Communications Magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[3] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G Networks: Use Cases and Technologies," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 55–61, 2020.

[4] ETSI, "Zero-touch Network and Service Management (ZSM)," ETSI GS ZSM 002 - Reference Architecture, Aug 2019.

[5] M. Fang, X. Cao, J. Jia, and N. Gong, "Local Model Poisoning Attacks to Byzantine-robust Federated Learning," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 1605–1622.

[6] C. Fung, C. J. Yoon, and I. Beschastnikh, "The Limitations of Federated Learning in Sybil Settings," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020.

[7] Y. Liu, X. Yuan, Z. Xiong, J. Kang, X. Wang, and D. Niyato, "Federated Learning for 6G Communications: Challenges, Methods, and Future directions," *China Communications*, vol. 17, no. 9, pp. 105–118, 2020.

[8] S. Savazzi, M. Nicoli, M. Bennis, S. Kianoush, and L. Barbieri, "Opportunities of Federated Learning in Connected, Cooperative, and Automated Industrial Systems," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 16–21, 2021.

[9] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed Federated Learning for Ultra-reliable Low-latency Vehicular Communications," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 1146–1159, 2019.

[10] J. Gallego-Madrid, R. Sanchez-Iborra, P. M. Ruiz, and A. F. Skarmeta, "Machine Learning-based Zero-touch Network and Service Management: A survey," *Digital Communications and Networks*, 2021.

[11] C. Benzaid and T. Taleb, "AI-Driven Zero Touch Network and Service Management in 5G and Beyond: Challenges and Research Directions," *IEEE Network*, vol. 34, no. 2, pp. 186–194, 2020.

[12] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine Learning with Adversaries: Byzantine tolerant Gradient Descent," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 118–128.

[13] R. Guerraoui, S. Rouault *et al.*, "The Hidden Vulnerability of Distributed Learning in Byzantium," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3521–3530.

[14] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust Distributed Learning: Towards Optimal Statistical Rates," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5650–5659.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.